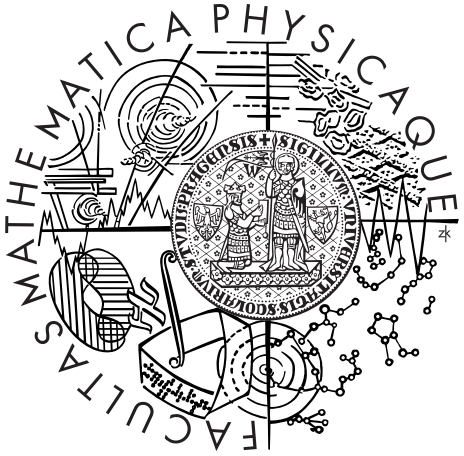
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Bc. Tomáš Helešic

Extrakce znalostních grafů z projektové dokumentace

Department of Software Engineering

Supervisor of the master thesis: Mgr. Martin Nečaský, Ph.D.

Study programme: Informatics

Specialization: Software Systems

Prague 2014

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date 1.4.2014 Tomáš Helešic

Název práce: Extrakce znalostních grafů z projektové dokumentace

Autor: Bc. Tomáš Helešic

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Martin Nečaský, Ph.D.

Abstrakt: S novými poznatky ve zpracování přirozeného jazyka a extrakce infromací z textu se otevírá možnost automatické extrakce znalostí a jejich sdružováním do znalostních grafů, které zachycují sémantické vztahy mezi těmito informacemi. Pro tyto grafy již existují úložiště a taktéž dotazovací jazyky, které uožnují přesnější a relevantnější vyhledávání oproti současným full textovým. Cílem této práce je prozkoumat možnosti automatické extrakce infromací z projektové dokumentace pomocí linguitického zpracování textů, zvolit vhodné datové uložiště a vybudovat nad ním vyhledávací službu.

Klíčová slova: Znalostní grafy, Extrakce informace, Zpracování přirozeného jazyka, Resource Description Framework

Title: Knowledge Graph Extraction from Project Documentation

Author: Bc. Tomáš Helešic

Department: Department of Software Engineering

Supervisor: Mgr. Martin Nečaský, Ph.D.

Abstract: With the new research progress in the natural language processing and information extraction from text, new possibility of automatic knowledge acquisition and its grouping into Knowledge graphs, that are catching the semantic relations between these entities is emerging. For these Knowledge graphs, data storages and also query languages already exists, which allow more precise and relevant search possibilities compare with current full text search engines. The goal of this thesis is to explore the opportunity of automatic extraction of information from project documentation with the use of linguistic text processing, design a proper data storage and build a search engine over it.

Keywords: Knowledge grahs, Information extraction, Natural language processing, Resource Description Framework

Preface

This research of my Master program has been done at Faculty of Mathematics and Physics, Charles University in Prague, in cooperation with ČEZ, join stock company. I gratefully appreciate many people who helped me at the project.

I would like to thank my supervisors at the Department of Software Engineering, Mgr. Martin Nečaský, Ph.D., Mgr. Jakub Kozák and my supervisor at the Instituted of Formal and Applied Linguistics, Mgr. Barbora Vidová-Hladká, Ph.D.. They led me through the though way of clarification and solving the goals of my Master Thesis.

I also want to express my gratitude to Mgr. Kiril Ribarov, Ph.D., who provided me with data and helped me with his outside view on the topic.

Contents

Introduction

For more then forty years, search has been about matching keywords to queries and for the search engine the keywords have been just words. The search engine could not assume anything about the words, their meaning and if they are related to other words. With the introduction of the Knowledge Graphs, Google started using the graphs to enhance their search engine with the semantic information about the objects and their relations stored in the graphs, the objects become things not strings. The purpose of Knowledge Graph is to catch important informations and relations. Interconnecting these graphs into larger knowledge base are allowing new generation of search engine, that tries to understand the world like people do[1]. The questions, that need to be answered are how to create the knowledge graphs and how to represent them. The creation problem stands for how to extract the informations and relations from texts. Most of informations are stored in unstructured or loosely structured documents, which makes the problem even more harder. If we manage to extract the informations from the document we need to represent them so the computer can manipulate and search over the. Every day more then several petabytes of documents are created and this huge amount of informations cannot be processed manually in the foreseeable future. To be able to process these huge amount of data we need to use automated information extraction based on the natural language processing.

Aim of the Thesis

The goal of this thesis is to explore the possibility of automated information extraction from unstructured or loosely structured project documentations into structured form based on principles of Linked Data[2]. The documents for the purpose of this thesis has been provided by the CEZ, a.s company. The information extraction (IE) will be based on the natural language processing (NLP), which are computational methods based on linguistic sound principles. The NLP will be done by a tool, which was created at the Institute of Formal and Applied Linguistics at Charles University in Prague and its called *Treex*[26]. The proposed solution will be implemented in a software module, which will enable user to set extraction patterns for entity and relation discovery, store the graphs in an appropriate data storage and will provide a search engine over these informations. The goal is to observe and test existing technologies, find a way how to interconnect informations extraction, natural language processing and company documentations. The last goal is to analyze designed solution, identify the most difficult tasks and proposed further development in this problematic. We can summarize the goals into 4 words: extract, represent, store and search.

Structure of the Thesis

The analysis and finding a suitable solution of the above described problems is covered in the first part of the thesis. It is consisted of six chapters that describes how the information can be extracted from the text based on the linguistic

knowledge. Precisely, for what we are searching in the results of NLP, how the NLP Treex can help us and how well we can extract informations from unstructured or loosely structured text. The chapters include IE, NLP, preprocessing of texts, processing results of NLP and the description of provided documents. They cover the problematic of finding optimal representation and database management system for storing extracted informations from texts altogether with the original and preprocessed documents. The last theoretical chapter describes the techniques of searching over the knowledge graphs, finding the appropriate query language and defining what types of quires we are interested in. The second part of this thesis covers the experimental implementation of the proposed solution. The last chapter of this thesis summarizes what we have accomplished and suggests further research in this topic.

1. What is Information extraction

This chapter explains what exactly information extraction is, the motivation why we are doing it and its origin. It clarifies from where are we extracting, for what are we looking for, what is the result and how can we use it.

1.1 Basics

The information extraction is any process which selectively structures and combines data which is found in semi-structured or unstructured texts. The strategy of the IE systems is to transform the text into informations that is more easily analyzed and digested. It isolates relevant text fragment, extracts relationships between them and then links this information into more coherent framework. It is being widely researched by the Natural language processing, Information retrieval and Web mining research groups. Two fundamental tasks of IE are relation extraction and name entity recognition. The name entity recognition is a process of finding entities such as people, organizations, tools, locations. The relation extraction refers to finding the semantic links between these entities in the text like `IsResponsible, Controls, IsManaging`. The evaluating process of extracting informations is concrete and can be performed automatically.[3][5][6].

1.2 Motivation

IE technology is reaching the market because of the great significance which can have to information end-user industries of all kind, such as finance companies, publishers and governments.

Lets start with an example,

Mike Ross and Rachel Stark are responsible for sales division.

we can extract the information from above sentence,

`ResponsibleFor(`*Mike Ross, sales division*`),`
`ResponsibleFor(`*Rachel Stark, sales division*`).`

The information shown in the example can be directly presented to an end user and what is more important it can be used by other systems as search engines and DBMS to provide better and more relevant services. The type and structure of the extracted information depends on the needs of an application. For example:

- Companies often need to manage their internal documents, e.g. project documentation, management documentation, reports. The key-goal is to provide more relevant search service over them, find hidden relations not only within one particular document but also between sets of documents. Managers often seek only for a relevant piece of information to help making their operational decisions. The reports contain informations about the competitors, supply level, undergoing repairs and events and they need

to get only the relevant parts from this report with removed redundant text. Such informations can be obtained by entity recognition and relation extraction.

- Researchers need to go trough a large amount of scientific publications to find some particular informations, genes, proteins, side effects of drugs and so on. The simple full-text search may not suffice because the entities have often synonyms and ambiguous names, making it difficult to obtain a relevant documents. The key point here is to identify mentions from the text and link them to their corresponding entities in an existing knowledge graph.

- Search engines become part of our daily life for most of the population, search based on matching the query to a particular string can no longer provide relevant result due to the enormous growth of data on Web. More sophisticated approach as entity, structure and question answering search can provide much better experience for the end-users. The IE stands here as a preprocessing step to improve the documents representations and to populate them into the underlying search engine's database.

If we would meet all the expectations based on the needs mentioned in the examples, the reliable IE system would make our life easier, safer and the business more successful. The goal of this thesis is to discover how much we can fulfill the expectations mentioned above from provided documents.

1.3 Analysis of IE results and their use in this thesis

Information extraction from texts dates back into the late 1970's, when the early days of NLP started[4]. The first commercial system was JASPER, it was developed in the mid-1980's by the Carnegie Group for Reuters with the goal of providing real-time financial news for traders[5]. In the 1987 the researched in the IE was encourage by the series of Message Understanding Conferences (MUC), which is a competition-based conference that was aimed on the following domains:

- MUC-1 (1987), MUC-2 (1989): Naval operations messages.
- MUC-3 (1991), MUC-4 (1992): Terrorism in Latin American countries.
- MUC-5 (1993): Joint ventures and microelectronics domain.
- MUC-6 (1995): News articles on management changes.
- MUC-7 (1998): Satellite launch reports.

The MUC was initiated by the DARPA, the US defense agency, with the aim to automate analysts processing of newspapers articles such as possible terrorists attacks. The IE in the early years of MUCs was about filling a predefined template with a predefined slots. Figure 1.1 shows a part of a template with the predefined slots used in MUC-4 and a sample part of the document which was processed to

Slot	Fill Value
Incident: Date	07 Jan 90
Incident: Location	Chile: Molina
Incident: Type	robbery
Incident: Stage of execution	accomplished
Incident: Instrument type	gun
Human Target: Name	Enrique Ormazabal Ormazabal
Human Target: Description	Businessman: Enrique Ormazabal Ormazaba
Human Target: Type	civilian: Enrique Ormazabal Ormazabal
Human Target: Number	1: Enrique Ormazabal Ormazabal
...	...

Extracted from:

Santiago, 10 Jan 90 Police are carrying out intensive operations in the town of Molina in the seventh region in search of a gang of alleged extremists who could be linked to a recently discovered arsenal. It has been reported that Carabineros in Molina raided the house of of 25-year-old worker Mario Munoz Pardo, where they found a fal rifle, ammunition clips for various weapons, detonators, and material for making explosives. It should be recalled that a group of armed individuals wearing ski masks robbed a businessman on a rural road near Molina on 7 January. The businessman, Enrique Ormazabal Ormazabal, tried to resist; The men shot him and left him seriously wounded. He was later hospitalized in Curico. Carabineros carried out several operations, including the raid on Munoz home. The police are continuing to patrol the area in search of the alleged terrorist command.

Figure 1.1: Example of the MUC-4 template for the description of terrorist attack

fill the template. Some slots were extracted directly from the text, for example *"Enrique Ormazabal"* and *"Businessmen"* however some slots had predefined values sets from which were appropriate words selected based on the information contained in the document, such as *"robbery"*, *"gun"*, *"accomplished"*. The task of filling the template is a complex work therefor the developers couldn't switch between various templates directly. This problem was well known so in the MUC-6 conference a number of independent template subtasks were defined. These tasks were consisted of named entity recognition, co-reference resolution, relation extraction and served as a building block to support domain-specific information extraction systems. All these systems developed in the MUCs are often defined as rule-based systems[7],[8]. These systems use linguistic extraction rule patterns developed by humans to match and locate information pieces in the text. The developed rules are strongly domain dependent, they can achieve good results however they are also labor intensive. All these limitations of manually developed rules forced researchers to start using statistical machine learning approaches. The decomposition of the IE systems into independent subtasks showed us that they can be turned into classification problems and classification problems can be solved with standard supervised learning algorithms as support vector machines, maximum entropy model. Hidden Markov models and conditional random fields for sequence labeling methods for identification segments of text have also been used. All these methods assume that the name entities, relations and template slots are well defined and are known in advance, however if we want to extract information from unknown sources the problem turns into unsupervised extraction. Entirely new way of IE problem is when the system is expected to extract all useful entities and relations between them from a large set of domains such as Web. The output of this system is a set of entities and descriptions of the relationships between them. The progress in this area includes systems like TEXTRUNNER[9],

WOE[10], REVERB[11]. Web information extraction systems are called wrappers and they rely more on HTML tags due to the fact, that the Web pages often contain tables, lists which are in structured and semi-structured blocks of text. But all this research done in the IE problem solving is more domain dependent with well known assumption of the text. But our goal is to find all potentially useful facts from a large and diverse corpus such as project documentations of a company. This is the goal of an open information extraction systems, which were introduced by Banko et al.[12]. They found that although different relation types have different semantic meanings, there exists a small set of syntactic patterns that cover majority of semantic relations. Therefore it is possible to create relation extraction model that can extract facts and key relationships between them from purely unstructured text with no assumption of what it contains[3]. And thats exactly what we need. To create open information extraction systems, which is domain independent and based on the syntactic patterns. The provided documents, which are described in the next chapter are unstructured and contains information from various domains. To reveal the syntax we have to use natural language processing tool, which will process the text of documents and give us results on which we need to apply the patterns to extract informations.

1.4 What is information

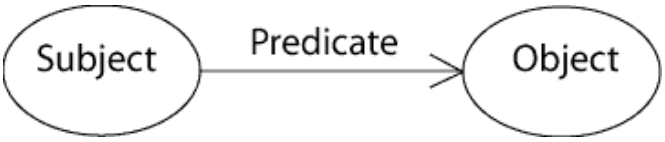
So far we have given an overview of historical and most recent approaches in the IE. But we need to focus on what we can extract from the text. The IE focuses on:

- Named entity recognition
- Coreference resolution
- Relationship extraction
- Semi-structured information extraction
- Language and vocabulary analysis
- Audio extraction

1.4.1 Named entity recognition and relation extraction

The task of the Named entity recognition (NER) is to identify real world objects from a text and to classify them into the set of predefined types like person, location, organization, medical drug and so on. The problem of NER cannot be accomplished by a single string matching against some sort of dictionary because the entity types do not form a closed set and usually are context-dependent. For example JFK stands for the name of the president or for the JFK international airport or any entity sharing the same abbreviation. The NER is mostly a preprocessing step before extraction more complex informations as relations and events. Early solutions of this task were using manually crafted patterns, later systems were trying to learn patterns from labeled data and the newest applications are using statistical machine learning for pattern discovery. Relation

extraction (RE) is about detecting and characterizing the semantic relations between entities in the text. The entities which are involved in the detected relation are called arguments. The first program that tried to extract binary relations was the Automatic Content Extraction program[13]. The boundaries for relation extraction are within a sentence, because it is very hard to create or automatically learn patterns for extraction relations between sentences. The researched done in Automatic Content Extraction project showed us, that we are able to work only extract information within the sentence. The system we desire is a system that is able to extract all information regardless of domain. The NER will be therefore transformed into simpler task of Named entity detection, which aims on finding entities without any knowledge about them. For example, *"Mike Ross is responsible for sales division"*, the named entity detection would denote finding *"Mike Ross"* as entity without having any knowledge about who or what it is but it detects that this sentence refers to *"Mike Ross"*. The same would be applied for *"sales division"*. The relation extraction then takes these two arguments and adds *"is responsible"* relation between them. A typical application of named entity detection and relation extraction is creating subject-predicate-object triples. The subject stands for the entity, predicate denotes the aspects of the subject and it expresses a relationship between the subject and the and the object that is another entity see Figure 1.4.1. Applied on the example sentence, subject is *Mike Ross*, object is *sales division* and predicate *is responsible*.



1.4.2 Co-reference resolution

This is more a linguistic problem of NLP. The coreference occurs when two or more expressions in the sentence or text refer to the same person or thing. Means that they have a same referent. *Mike Ross suggested that he would like to manage accounting*, the noun *Mike Ross* and pronoun *he* refer to the same entity. We have to rely on the NLP tool to recognize the coreference and provide us with the correct referent. Otherwise we have to create specific pattern to recognize the coreference in the results of a linguistic processing.

1.4.3 Semi-structured IE

Semi-structured IE deals with tables and tries to extract the information from them. There are various techniques how to do that. One is based on knowledge what kind of information rows and columns contains. It means to create hand written rules which turns out to be useless for automatic extraction from various tables. However if we would know that these tables structure do not change, we would able to extract relevant pieces of information without any significant loses. Another approach is to transform these tables into textual form and extract the information from the text as any other plain text. However to create such preprocessing seems to be an extremely difficult task for NLP researchers.

Another part of semi-structure IE is to process comments under articles, it tries automatically to find correspondence between these comments and parts of the text on which they are pointing. It seems to be also an interesting part however not useful for this thesis.

1.4.4 Language and vocabulary analysis

The goal of this task is to automatically extract relevant terminologies from a given domain. We can imagine it as an identification of key words and multi-word phrases uniquely describing certain domain. The domain is relatively large, it could be some web community, enterprise or something more concrete as set of documents. Terminology extraction is typical for web applications, like web crawlers[15], recommender systems[16] and many others. The extraction process uses NLP, it searches for noun phrases (NP), adjective-NP and prepositional-NP. The set of these terminologies is then filtered trough statistical and machine learning methods to create low ambiguity and high specific subset. However to filter effectively we need again to have some knowledge about the domain, but do we really need to filter these terminologies? What about using the terminology extraction to create a terminology vocabulary for each document. We then could count the frequency of the terms and overall frequency over the whole set of documents. Then we would be able to identify the documents that are containing unique terms and also to identify which of them are describing the same domain and group them.

1.4.5 Summary

We have not described the Audio extraction, since we are only dealing with textual documents and this does not have any importance for us. We have given a list of key parts of IE. We need now to look into the documents and find a way how to apply and test these task on them. However these problems cannot be done without using a NLP tool, because we need to discover the syntactic structure of sentences together with other morphological information about the words that they contains to be able to solve the tasks. The next chapter will give a detail description of received documents.

2. Description of the provided documents

In this chapter we will give an overview of documents that we received from the CEZ, a.s company. Precisely we will describe the scope of the documents, their size, structure and format. In the last section will try to identify and select the most interesting ones on which the research of this thesis will be based.

2.1 Documents

The documents are separated into 13 different sets. Each of them contains various types of company documentations: presentations, reports, records, employees profiles, project documentations. The following subsections describe every set.

2.1.1 Field of experience summary

This set contains documents, summarizes knowledge management and it is consisted of 32 files. Most of the documents are PowerPoint presentations. These presentations have many tables, enumeration, short sentences and abbreviations. Sometimes they contain inner documents of Word and Excel type. Other documents in this set are exported probably from database and accommodate experience description of various events. Each of this documents contain more then one of these events. The presentations have mostly more then 20 slides. The length of the inner word and excel documents vary from 1 page up to 10.

2.1.2 Experience records

This is one of the largest set and has about 250 files. The documents are mostly describing individual records of experiences, observations and researches. The first page summarizes the document in a table. The table contains names of the authors and supervisors, short description of the event, keywords, date of origin and many other details. The text is structured in sections with long continuous texts, tables, enumerations. All these documents were created in the Microsoft Word and have mostly more then 5 pages. Some of the documents are also in Portable Document Format *(PDF)* and were scanned from a printed version of these Word files. Other documents in this set are various forms and excel tables with names, numbers, shortcuts and phrases.

2.1.3 Expert's profiles

All documents are PowerPoint presentations with the length of one or two slides. The slide is consisted from one big table containing employee's profile, probably imported from a database. The profiles store informations about name, ID, position, experience, role, education, references and many other details about employees. There are 13 documents in this set.

2.1.4 Substitution program

This set of documents is about internship program within the company. The documents are reporting the transmission of expertise and experience between employees. The documents are divided into groups of Word document files, excel spreadsheets and PDF files. Word and PDF files have mostly structured introduction describing the goals of the educational program followed by detail report in multiple pages. The Excel spreadsheet documents are forms with predefined slots filled with description about the program event. 59 documents in this set.

2.1.5 Knowledge management concept

It has only one file, Power point presentation exported into PDF format. The document has 124 slides with many tables, indented short texts, images. The purpose of this document is to present the knowledge management concept.

2.1.6 Forms

The documents are combination of forms and templates for writing reports and events of various projects. It is a mixture of Word documents and Excel spreadsheets with short coherent texts and it is structured into tables with pre-filled example values. The set has 17 files.

2.1.7 Yearly reports

Presentations, transcripts and documentation of annual projects' reports. Presentations are filled with tables, images, itemizations with short sentences. Transcriptions contain structured list of goals with short descriptions. Documentations are mostly divided into sections with structured longer continuous text spanning into several pages. 10 files in this set.

2.1.8 General presentations

PowerPoint presentations of various projects, events, educations. The structure and length of this presentations vary in every document. They are consisted of tables, itemization, images, no larger continuous text.

2.1.9 Reports

3 Excel spreadsheet reports. Tables with dates, names, values, phrases. No continuous text is presented.

2.1.10 Experience records from KM

Experience records are same as documents in section 2.1.2 but from a different source. All documents are in PDF format exported from the Word document. The set contains 9 large documents.

2.1.11 Management documentation

Documentations of various project written for the needs of management. Textual documents with a summary in the first page followed by sections with longer continuous text. The text is divided into sections containing tables and itemization with texts of various length.

2.1.12 General enterprise documentation

This set of documents is a mash-up of presentations, educational texts and tests used to prepare employees, suppliers and emergency staff to work in restricted areas of the company. The set contains about 300 different files. Every document differs in content, size and structure.

2.1.13 Supplier's training

This set accommodates presentations about training programs for suppliers to be able to work in the CEZ company. The use of tables, itemization, images are typical for this type of documents. There is about 50 files in this set.

2.2 Summary

The received files represents typical daily company documents. The common types of the files are Word, Excel, PowerPoint, PDF. Some of the PDF files are printed from other documents, however some of them are scanned that will make text extraction difficult. The purpose of this thesis is to test the possibility of the information extraction, natural language processing and semantic searching. It implies that we need to create a representative set from these documents. Therefor we need to select every type of these documents, but not only the different types of the files as .doc, .pdf, .ppt, but also the different contents. We need to have a set containing documents with large continuous texts, itemized short texts, tables, texts with abbreviations, phrases. This set will be created from these documents:

- **Experience records** - semi-structured large texts with tables, abbreviations, values. Different domains of information. Word and PDF types.
- **Expert's profiles** - structured PowerPoint files with short itemized texts, abbreviations.
- **Substitution program** - reports, different file types, different domains, larger texts.
- **Management documentation** - project documentations, larger texts, different domains.

Therefore we will now be working only with these documents.

3. Natural language processing

In this chapter we will be talking about the Natural language processing. We will begin with a motivation in the context of our goals. Then we will continue to the linguistic processing of texts and what are results of this process. We will describe the NLP tool Treex developed at the Institute of Formal and Applied Linguistics at Charles University in Prague. In the last section we will exploit the dependency tree and what kind of informations these trees contains.

3.1 Motivation

The natural language processing (NLP) is a field of computer science, AI and linguistics that explores how computers can be used to understand and manipulate with natural language text or speech. The first steps were made by the Georgetown experiment in 1954 which involved fully automatic translation of more then sixty Russian sentences into English. The expectations were that within 5 years the machine translation will be solved, however the task revealed itself to be much more complex and till the 1980's the fundings in this field of computer science were drastically reduced. The resurrection of NLP came with the first introduction of statistical machine learning, before that the NLP used only hand written rules and for that reason the usability was limited only for a certain domains. The statistical machine learning introduced general learning algorithms to automatically learn rule trough the analysis of large corpora, mostly newspaper articles as typical real-world examples[19]. The major tasks of NLP are:

- Machine translation
- Word segmentation, sense disambiguation, morphological segmentation, parsing
- Question answering
- Information extraction with subtasks relationship extraction and named entity recognition
- And many others

Machine translation and question answering are out of the scope of this thesis. Word segmentation, sense disambiguation, morphological segmentation, parsing are all part of the linguistic text processing. They are all chained in a pipeline which processes a text and provides us with linguistic information about sentences and words. The goal of this pipeline is to make formal analysis of a sentence resulting in a dependency tree showing the syntactic relations in the sentence, together with morphological informations about words such as word constituent, lemma. With the knowledge of a structure of a sentence together with morphological informations about its words we can then start searching for a specific patterns in the sentences and gather informations in which we are interested in[17]. But before searching for patterns we need to understand how the linguistic processing works exactly and what kind of informations in detail we

could obtain by processing our documents. After we will understand results of NLP we can continue to IE.

3.2 Analysis of Natural language text processing

There are many tools which provide linguistic processing of natural text. For example Treex[26], OpenNLP[27], Stanford CoreNLP[28]. A text is processed in a pipeline, it means each layer takes input from previous layer and sends its result to the next layer. The typical pipeline is shown on the figure3.1. The text enters the tool in a plain text format and therefore we need to extract text from various document types into the plain text. First layers is a *Sentence detector*, it detects and breaks down the text into sentences. It passes a set of sentences to the *Tokenizer*, which splits the sentence into tokens, that are words, numbers, delimiters and abbreviations. The next layer is a *POS Tagger*, *POS* stands for part of speech tagger. The tagger takes tokens and annotates them with morphological tags and morphological lemma. A morphological lemma is a word's basic form or normalized form and it matches the unique key of the corresponding entry in the morphological dictionary. The morphological tags are consisted of word class, person, tense, if a word is in negation and many more. Another part of the tagger process is disambiguation. A word can have different meanings and disambiguation determines the correct one. The last layer is *Treebank Parser*. It processes a sentence and creates rooted ordered node tree with labeled nodes and edges. One token of the POS tagger is represented exactly by one node in the tree and the dependency relation between two nodes is captured by the edge between these two nodes. The actual type of the relation is given as a functional label of the edge. Most of the edges represent dependency relations, while others mirror various linguistic or technical phenomena, e.g. coordination, apposition, punctuation. Linear ordering of the nodes, which corresponds to the original sentence word ordering is also recorded, allowing correct graphical rendering of the tree. Some tools provide additional layers but they are not important for this thesis. For us the NLP ends with the Treebank parser. We have all linguistic information about a sentence that we need. Please notice that the NLP works only with sentences, it does not provide any information whether two sentences have a same subject or not. We are now ready to start searching for a patterns which we need for IE[21].

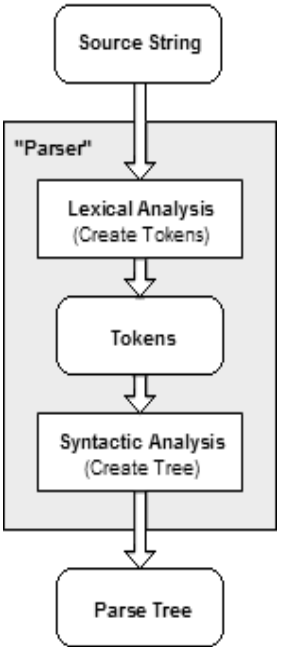


Figure 3.1: Linguistic pipeline

3.3 NLP tool Treex

We will now provide an overview about the linguistic tool *Treex*, formerly (*TectoMT*) developed at the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics at Charles University in Prague. Its a highly modular

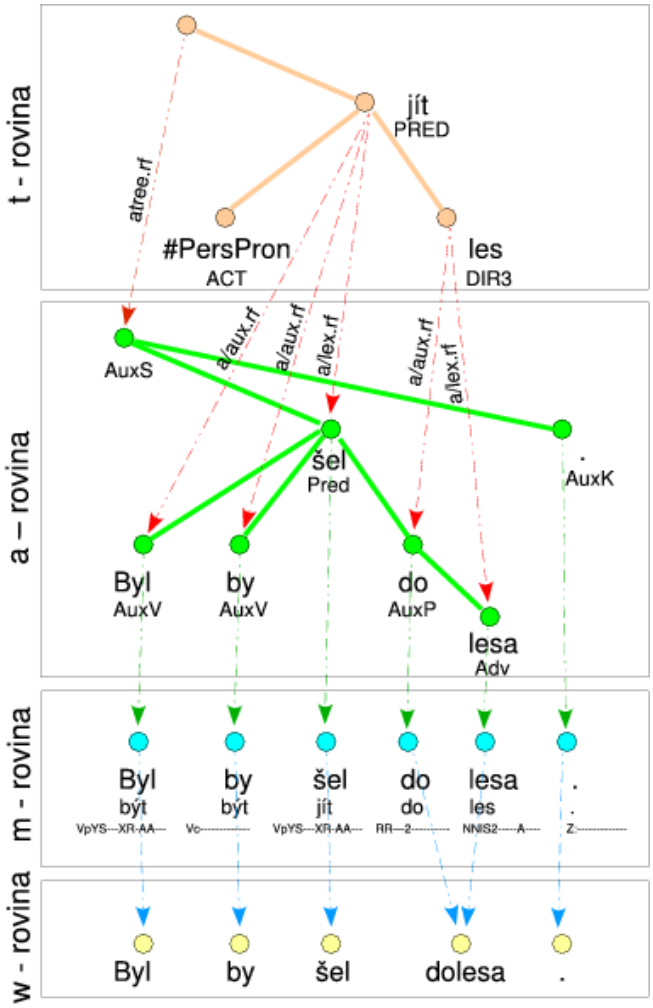


Figure 3.2: Layers of language description. Image taken from [22]

NLP software system implemented in Perl on Linux platform. It is primarily aimed at machine translation, but due to re-usability of the numerous integrated processing modules (called blocks), which are equipped with uniform object-oriented interfaces it can be used for other NLP tasks such as information extraction. Treex provides standard NLP functions such as tokenizer, lemmatization, tagging and parsing. The framework profits from stratification language approach, it defines four layers of language description (developed for the purpose of Prague Dependency Treebank[21]) listed in the increasing level of abstraction. The layers are shown in the figure 3.2.

- word w-layer represents raw text, it splits the text into sentences and tokenizes them.
- Morphological layer m-layer, each sentence is tokenized and annotated with a lemma and a morphological tag. Details [23].
- analytical layer a-layer, each sentence is represented as a shallow-syntax dependency tree. Each node is annotated with constituent and dependency relation to its parent. Details [24].

- tectogrammatical layer t-layer, represented as syntax dependency tree containing only meaningful words. Auxiliary verbs, prepositions etc. are not presented. Details [25].

Following the assumption that every non-trivial NLP task can be decomposed into sequence of steps, these steps are implemented as reusable components called *blocks*. Each block has defined input and output format specification and linguistically interpretable functionality. These block are then put into the scenario which creates a pipeline for processing the text. Example scenario:

Scenario A
`Read::Text`
`W2A::CS::Segment`
`W2A::CS::Tokenize`
`W2A::CS::TagMorce`
`W2A::CS::ParseMSTAdapted`
Write::CoNLLX

Application represents an end-to-end NLP task, which is described by the scenario. It starts with the *reader* as input conversion, processing the blocks and ends with *writer* as output conversion. There are readers and writers for a number of formats: plain text, CoNLL, PDT PML, Tiger and many others.

3.3.1 Running Treex

Lets demonstrate Treex on and example:

"Pražský závislostní korpus vychází z dlouhodobé pražské lingvistické tradice, upravené pro současné potřeby výzkumu v oblasti komputační lingvistiky."

Having this sentence stored in *example.txt* file we can start processing the document by invoking Treex from the Linux terminal:

```
treex -Lcs Read::Text from="example.txt" W2A::CS::Segment
use_paragraphs=0 use_lines=1 W2A::CS::Tokenize
W2A::CS::TagMorce W2A::CS::ParseMSTAdapted
Write::CoNLLX to="exampleInConllx.txt"
```

With the argument *-Lcs* we are telling Treex to set the environment for the Czech language. *Read::Text* sets the *Reader* to process plain textual input, the following argument tells which file we would like to process, otherwise the Treex is waiting to receive the text on a standard input. The flags *use_paragraphs=0 use_lines=1* tells the segmentation process that the empty line does not mean the end of a sentence and there might be more then one sentence on a line. *W2A::CS::Tokenize* stands for use of tokenizer for the Czech language. *W2A::CS::TagMorce* sets the use of the POS Tagger MORCE[29]. *W2A::CS::ParseMSTAdapted* says that we would like to parse the sentences using *Minimum-Spanning Tree Parser*[30]. Due to the highly modularity of Treex we could use different tagger or parser such as *ParseMSTperl* or others. Finally *Write::CoNLLX* tells Treex to convert the output into CoNLLX data format[31]

Table 3.1: ConLLX result format

ID	Word	Lemma	CPOSTAG	POSTAG	FEATS	PHEAD	DEPREL
1	Pražský	pražský	_	AAIS11A	_	3	Atr
2	závislostní	závislostní	_	AAIS11A	_	3	Atr
3	korpus	korpus	_	NNIS1A	_	4	Sb
4	vychází	vycházet:T	_	VBS3PAA	_	0	Pred
5	z	z1	_	RR2	_	4	AuxP
6	dlouhodobé	dlouhodobý	_	AAFS21A	_	9	Atr
7	pražské	pražský	_	AAFS21A	_	9	Atr
8	lingvistické	lingvistický	_	AAFS21A	_	9	Atr
9	tradice	tradice	_	NNFS2A	_	5	Adv
10	,	,	_	Z:	_	11	AuxX
11	upravené	upravený(*3it)	_	AAFS21A	_	9	Atr
12	pro	pro1	_	RR4	_	11	AuxP
13	současné	současný	_	AAFP41A	_	14	Atr
14	potřeby	potřeba	_	NNFP4A	_	12	Adv
15	výzkumu	výzkum	_	NNIS2A	_	14	Atr
16	v	v1	_	RR6	_	11	AuxP
17	oblasti	oblast	_	NNFS6A	_	16	Atr
18	komputační	komputační	_	AAFS21A	_	19	Atr
19	lingvistiky	lingvistika(věda)	_	NNFS2A	_	17	Atr
20	.	.	_	Z:	_	0	AuxK

and store it in a file, otherwise it would print it on the standard output channel. Lets look at the result:

As you can see in the result3.1 of the *Treex*, each line corresponds with exactly one token from the original sentence and it contains 8 different fields separated by the tabulator character. Each sentence is separated by an empty line. The most important fields for us are:

- ID - Token counter, starts from 1 for each new sentence.
- FORM - Word form, or punctuation symbol.
- LEMMA - Word lemma. The lemma is sometimes followed by information related to word disambiguation.
- POSTAG - Part of speech tag. *Treex* uses 15 position tag from Prague Dependency Treebank 2.0[33].
- HEAD - Projective head of current token, which is either a value of ID or zero if the token is root node or an underscore if it is not available. In other words it is a link to a parent node in a dependency tree. It determines the dependency of the word.
- DEPREL - Dependency relation to the PHEAD, or an underscore if it is not available.

The full documentation is available here [32]. The POSTAG and DEPREL differs for each language. The result can be easily parsed to create our own in-memory representation of the sentence and apply extraction on it.

3.4 What carries information in a sentence

Before we advance furthermore we need to find out what is carrying information in the sentence. As we have mentioned the IE is based on the parsing of sentences and it applies patterns to search in the dependency tree for information. We need to know the linguistic background which constituents in the sentence are important and why. The explanation will be made for the Czech language and we will use the constituent tags used in *Treex* to get to know them before we advance to the next chapter, which will be about creating dependency patterns for IE.

The parser in *Treex* distinguishes 28 different analytical functions[34]. These functions describe the semantical relations between two nodes in the dependency tree. Precisely in which relation stands a child node to a parent node. There are 5 groups of linguistic phenomenon in which these function appears and are typical for them. We will go trough them now.

3.4.1 Simple sentence containing verb

Simple sentence[35] which contains a *Predicate* (verb) is the most common sentence structure in the Czech language. The predicate is a root node in the dependency tree and it is a basic constituent that is attributing someone or something a feature, state, change or activity. Pay attention to the feature of this constituent ättributingẅhich means that the predicate is giving two or more arguments in the sentence into a relation described by the verb. Can you see the correspondence between the description of the predicate and the description of relation extraction from the IE? And that is exactly for what we will be searching in the results of *Treex*. All we need now is to find what are these arguments. One of the argument is always a *Subject* if it is directly presented in the sentence. However subject can be sometimes unexpressed and then it is hard to find to which subject in surrounding sentences is related. The subject is mostly expressed with a noun but it can be also expressed by any language construction on which we can ask *Who* or *What*. The subject is an originator of an event, carrier of some act, state or feature. We have now *Subject -¿ Predicate -¿ missing argument*, we need to find who is the last argument. A Predicate can be developed with *Object*, *Complement* and *Adverbial*. The object describes what is the result of the act in a sentence *write letter*, or what the act is directly affecting *(touch the table)*, or towards what the act it is pointing *(advising a boy)*. The complement is dependent on two constituent at once one of the complements is mostly predicate the other constituent can be subject or something else *(The boy was lying ill)*. We will focus only on cases where it is dependent only on a predicate. The last constituent which can develop predicate is an adverbial. The adverbial specifies the circumstances and relations as location, time, way, comparison, rate, instrument, cause, effect or purpose. These 3 constituent are last arguments for that we will looking for. We now know how to recognize the originator, the attributer and the result in a sentence. The other constituent which is commonly presented in a sentence is an *Attribute*. It depends on the noun of any constituent in the sentence and closely determines and defines its expression. The last constituents are *Auxiliary constituents* and *Punctuation*. Not important for the IE.

3.4.2 Ellipsis

Ellipsis[36] is a sentence with a missing expression, mostly missing predicate. In a written text this phenomenon appears usually in a addresses, surveys, forms and tables. If we consider extracting text from these elements we have to deal with a degenerated dependency tree. Degenerated means that the important constituent is missing and all constituents which would dependent it get the tag *ExD*. If we are lacking predicate in the sentence, and subject with object is presented, they will not be marked with the right function but they will get the *ExD* and therefore we are unable to extract the full information from the sentence. Still we can determine if the words are noun or other constituents. These *ExD* can be developed by other child nodes, for example by attributes.

3.4.3 Coordination

Coordination[37] is a relationship between two or more constituents in two or more main or subsidiary sentences. In the coordination relation can be practically any constituents, as subjects, predicates, objects and others. They are marked with *-Co* tag after the functional tag. In the coordination can be only same constituents. These constituents in the relation has parent node primarily conjunction as comma. Lets have a example sentence:

"John works in Boston and in Prague."

The subject *John* depends on the predicate *works*. We have two objects in the sentence *Boston, Prague*, these objects are in a coordination and are dependent on the conjunction *and*. We need to extract these object arguments and connect them in the predicate relation with the subject argument.

3.4.4 Parenthesis

It is a supplementary added comment[38] to an expression in the sentence. The originator in the sentence gives an additional explanation, the expression is marked with the *-Pa* tag. The parenthesis needs to be separated from the main sentence with parentheses, dashes or commas. If this condition is not met the constituents will not get this tag. We need to process this phenomenon in a same way as coordination.

3.4.5 Complex phenomenons

Complex phenomenons[39] are anomalies which contains different categories. The typical example is a direct speech sentence. The Direct speech in a linguistic reproduces what the originator has said directly, literally. The first part of the sentence is introducing who, in which way and in which circumstance the originator has said the speech. This phenomenon is really complex and its hard to precisely create a dependency tree and extract the information stored in it.

4. Information Extraction

We have a full overview about the documents, how linguistic text processing works and what results it can offer us. In the chapter 1 we have discussed general IE task and we have explored how we could apply them on our goal. In this chapter we will analyze existing patterns for information extraction in correspondence with the Czech language. We will focus on entity detection, relation extraction and we will end with terminology extraction.

4.1 Dependency patterns

Many techniques of the automated acquisition of IE systems have used dependency trees to extract places of interest from the text[17]. These systems used variety of patterns models, which are based on deep studying of the dependency analysis. The IE system searches for the patterns in q dependency tree, if a match has been found, then it extracts the words or short phrases from this match and processes the information based on the needs of the systems, that means relation extraction, entity recognition/detection and terminology extraction. Any suitable model should be expressive enough to represent particular information pattern without being too complex. We will now introduce several pattern models used for relation extraction and named entity recognition, for demonstration purposes we will use as an example following sentence (taken from [18]) with its' corresponding dependency tree on figure 4.1. *"Acme Inc. hired Mr Smith as their new CEO, replacing Mr Bloggs."* The example describes several relations and entities as between person and their employer *(Mr. Smith -Acme Inc., Mr. Bloggs Acme Inc.)*, a person and their job title *(Mr. Smith - CEO, Mr. Bloggs - CEO)* and two people changing jobs *(Mr. Smith - Mr. Bloggs)*.

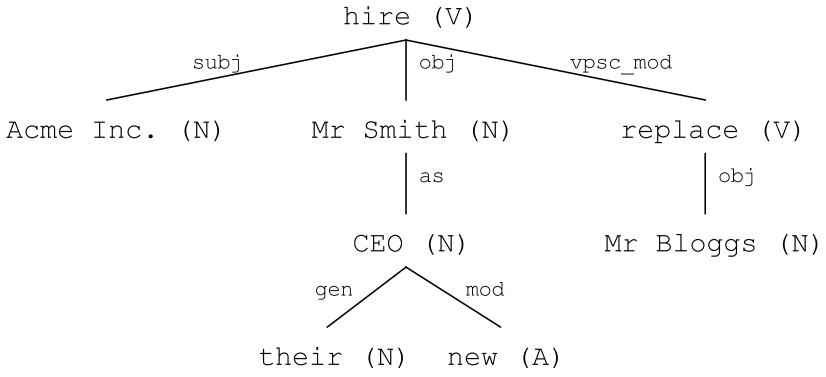


Figure 4.1: Example of dependency tree

There has been several attempts to measure the expressiveness and performance on a large corpora of text including newswire stories and newspaper articles. The complexity of the models is measured in terms of possible patterns which they can generate on the dependency tree not in terms of complexity in implementation of the models into the IE systems.

4.1.1 Predicate-Argument model (SVO)

Firstly introduced by Yangarber (2003), Stewenson and Greenwood (2005), the extraction pattern consists of subject-verb-object tuples. The pattern has a verb as a root and direct subject and/or object. The pattern is extracted from every sentence which contains a verb it cannot extract any information in phrases (non-verb sentences). From the example following tuples will be extracted:

```
[V/hire](subj[N/Acme Inc.]+obj[N/Mr Smith])
[V/replace](obj[N/Mr Bloggs])
```

The model ignores prepositions and other constituents and the restriction on direct successors of the verb might be too strong. For example the relation between the person *Smith* and the job title *CEO* could no be extracted. However the model is very simple and can extract the most significant relations and entities.

4.1.2 Chain model

The Chain model defines the pattern as the direct path between a verb and any of its descendants. Eight chains can be extracted, for example:

```
[V/hire](subj[N/Acme Inc.])
[V/hire](obj[N/Mr Smith])
[V/hire](obj[N/Mr Smith](as[N/CEO]))
[V/hire](obj[N/Mr Smith](as[N/CEO](gen[N/their])))
```

The model provides a step beyond the direct arguments of predicates and includes parts of the dependency tree which would be ignored by the SVO model, but this model cannot represent a link between arguments of a verb *Acme hire Smith*.

4.1.3 Linked Chain model

The models represents extraction patterns as a pair of chains which share the same verb as their root but do not share any direct descendants. This model can extract much more information from the sentence due to the possibility of linking together event participants compare to SVO and Chain model. This model generates 14 patterns from the example sentence such as:

```
[V/hire](subj[N/Acme Inc.]+obj[N/Mr Smith])
[V/hire](subj[N/Acme Inc.]+obj[N/Mr Smith](as[N/CEO]))
[V/hire](obj[N/Mr Smith]+vpsc mod[V/replace](obj[N/Mr Bloggs]))
```

4.1.4 Unconstrained Linked Chain model (ULC)

In the Linked Chain model, the pair of chains have to have verb as a root, the ULC model removes this restriction and allow patterns, which are rooted at any node. It increases the complexity and from the example sentence, it generates 32 patterns including:

```
[N/CEO](gen[N/their]+mod[A/new])
[V/hire](subj[N/Acme Inc.]+obj[N/Mr Smith](as[N/CEO]))
[V/hire](obj[N/Mr Smith]+vpsc mod[V/replace](obj[N/Mr Bloggs]))
```

Model	Parser minipar	Stanford	MALT
SVO	16,915	11,271	11,528
Chain	196,887	208,063	302,559
Linked chain	715,503	788,542	1,268,676
ULC	845,429	1,079,659	1,657,934
Shortest path	1,194,910	1,604,426	2,326,893
Subtree	1.64×10^{64}	1.70×10^{12}	4.56×10^{16}

Figure 4.2: Number of patterns generated by the models

4.1.5 Shortest Path model

This model allows a pattern to be formed by following the shortest path in the tree between any pair of nodes as well as simple chains and ULC chains. 28 shortest path patterns can be generated including:

```
[V/hire](obj[N/Mr Smith](as[N/CEO]))
[N/Mr Smith](as[N/CEO](mod[A/new]))
[V/hire](subj[N/Acme Inc.]+obj[N/Mr Smith](as[N/CEO]))
[N/CEO](gen[N/their]+mod[A/new])
```

4.1.6 Subtree model

The last presented model generates 43 patterns from the example sentence. In this model any of the subtree of the dependency tree can be used as an extraction pattern, where the subtree is formed by any connected subset of nodes. The subsets containing only single node is note to be considered as pattern. Its the most richest representation of the dependency tree, it generates all the patterns discussed in previous models. However it seems to be too difficult to select from this pattern the subset with the most information gain.

4.1.7 Patterns comparison

The models were measured and evaluated to discover how much of the target information each could capture on a large corpora containing biomedical text and newswire articles[18]. Mark Stevenson and Mark A. Greenwood used variety of dependency parsers (MINIPAR, The Standford parser and MaltParser) as well as annotated corpus from the Sixth Message Understanding Conference (1995) and training corpus used in the LLL-05 challenge task (2005). The measurement was consisted from total number of generated patterns and relation coverage. The figure 4.2 shows the variety of generated patterns by this models. This diversity in values can be explained how accurately the models processes the natural text. As can be seen, the accuracy of the NLP parser determines the number of generated patterns and this accuracy cannot be underestimated. More accurate parser provides more accurate dependency trees for the models enabling them to generate and cover more relations and entities.

		SVO		Chain		Linked Chain		Comprehensive
Parser	Corpus	%C	%B-C	%C	%B-C	%C	%B-C	%C
minipar	Management	7.49	9.07	41.07	49.73	76.78	92.95	82.60
	Biomed	0.93	1.30	17.38	24.44	62.53	87.93	71.11
	Combined	3.14	4.19	25.39	33.86	67.35	89.81	74.99
Stanford	Management	15.05	15.10	41.07	41.20	93.65	93.93	99.70
	Biomed	0.46	0.49	16.53	17.39	88.53	93.13	95.06
	Combined	5.40	5.58	24.83	25.69	90.26	93.42	96.62
Malt	Management	5.98	6.61	33.96	37.57	79.35	87.87	90.39
	Biomed	0.23	0.26	11.63	13.07	73.04	82.11	88.60
	Combined	2.17	2.43	19.18	21.44	75.17	84.05	89.44

Figure 4.3: Coverage and bounded coverage of models

The second measurement examined whether the generated patterns cover the information which should be extracted. The pattern covers a relation if it contains the two items which participate in the relation. Generally speaking it compares the number of extracted relations and entities against the annotated corpus. The parser cannot process accurately every sentence in the text and for that reason they added a third measurement, bounded coverage. The bounded coverage removes from the corpus relations that cannot be processed by the models due to the false dependency parsing. Figure 4.3 displays the models performance in coverage (C) and bounded coverage (B-C) test. The measurement was performed only on the SVO, chain, and linked chain models, because the UCL, shortest path and subtree model does not need a verb to create a pattern and can generate patterns from non-verb sentences and in bounded coverage test achieved 100% coverage. The simplest model SVO, does not performed well in combined corpora with less then 6% coverage. The reasons are, that the information are described in the way in which SVO model is unable to represent due to the direct successor constraint. The Chain model covers a greater percentage of the relations, however with still less than half of the relations. The best results achieves linked chain model with 80%-90% of coverage. The measurement indicates that the SVO and chain models are extracting with severe losses compare to linked chain model. However we need to ask ourself, what kind of entities and relations we would like to extract and if the combination or some altering of them could serve our cause enough.

4.2 Analysis of patterns and possible solutions

We have described a list of existing patterns used for IE extraction together with the comparison of effectiveness on biomedical and newswire articles. However it seems that all we need now is just to apply these patterns on our documents and we are done. Its not that simple because these models are extracting nodes without considering if the nodes are containing relevant knowledge. What we need now is to go trough the tasks of IE named entity detection, relation extraction, terminology extraction, look into the syntax of Czech language, in the results of NLP and define new or alter already described patterns that will serve our goal well.

4.2.1 Named entity detection and relation extraction

As we have spoken in previous chapters Named entity recognition needs to have a knowledge about the domain from which we are extracting. In our case this prerequisite cannot be met. Is it different with the entity detection? We know that some constituents are entities which are carrying information we need to find. Subject is an originator of an event in the sentence. Objects, complements and adverbials are developing a predicate in a sentence and are carrying information about a result of an act or explaining and more closely specifying the subject and its feature, act or state based on the predicate. So if we can create rules and specify that subjects, objects, complements and adverbials are important for us, than we can extract these entities from the text and actually get the word expressing this constituent and the information that contains, figure4.4. The rule will be simple:

1. Find all predicates, subject, objects, complements, adverbials in the sentence
2. For all subjects (in a complex sentence can be more then one subject) find on which predicate are dependent and create a joined them. Only directly dependent
3. For all object, complements and adverbials find on which predicate are dependent. Only directly dependent
4. For all joined doubles from step 2 and 3 joined them on common predicate

This will ensure us that we will extract the originator of the sentence and the affect of the originator and predicate. Extracting relevant key informations from the sentence.

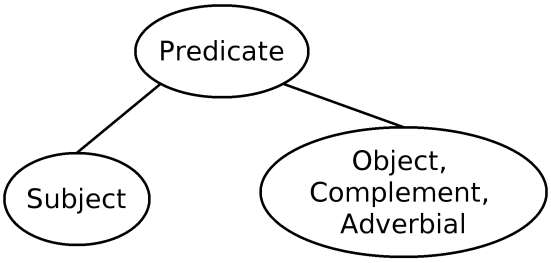


Figure 4.4: Pattern subject-predicate-object

4.2.2 Co-reference resolution

Co-reference resolution is a more advanced task in NLP . The goal is to find all expression that refer to the same entity in the text. This task can be solve in several ways. First option is to create a specific patterns to search in the dependency tree for this occurrences. Since the entities in coreference relation can be different constituents, the number of patterns will be high. Other option is to use the tectogrammatical layer from PDT, which identifies can identify the coreference. The tectogrammatical alters the dependency tree from analytical layers and removes non important nodes and only nodes with semantical meanings

remains. Although the tectogrammatical tree would be easy to process, *Treex* does not provide this option in processing the text. We can work only with the dependency tree. Its seems not promising to catch the coreference.

"The music was so loud that it could not be enjoyed."

The subject *music* and the pronoun *it* are in coreference relation and we should consider post-processing after we extract the entities from the text that will try to bind the entities together and replace the pronoun with noun subject. However this could be done only within the sentence. Co-reference resolution is too complex task and does not have any similarity with patterns described above.

4.2.3 Coordination

As mentioned in previous chapter constituents can be in coordination meaning that they are of the same type and they depend transitively on same constituent in the sentence however the direct parent node is comma or other punctuation. We need to transitively find if the transitively parent constituent is predicate, but no other constituent except the punctuation is on the way to the predicate. If we could solve this, we can extract the information in section4.2.1. The algorithm described there has to be altered to not only use direct dependency but take into account also the transitive dependencies, see figure4.5.

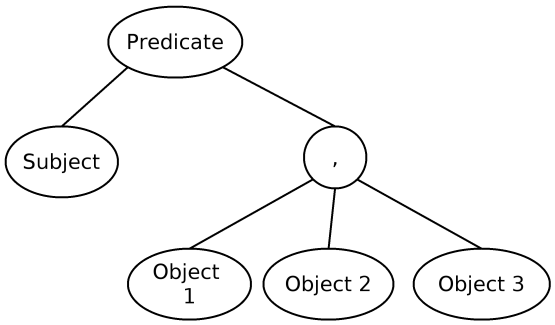


Figure 4.5: Pattern for transitive dependency

4.2.4 Terminology extraction

Terminology extraction is another large task in NLP. There are many ways how we can extract the terminology. One ways is trough linguistic processing the other way is without it. Without the linguistic processing we usually need to have a language corpus with word counts. When we process the document, we count the frequency of words in the documents. Then we compare this sum with the frequency of the word in the language. If the word is frequently repeated in the document but not in the corpus, it is probably a term. There can be many other algorithms or techniques how to extract the term, but since we are using linguistic processing tool we will try to create our own terminology extraction. The simplest method is to consider every noun to be a term. Noun from linguistic perspective is a part of speech that marks names of persons, animals, things, features and acts. The number of extracted nouns could be really high, thats why we need somehow to select only the significant ones. Since we have the dependency tree with constituents, we can try to search for a specific constituents (entities) which

are nouns and filter those nouns which are not carrying any semantic meaning in the sentence. But we do not have to stop here, we can also use the edges going from an entity in the dependency tree since we know that these edges means dependency relation and are developing the parent entity. One of these constituents is *Attribute*, which depends on the noun and closely specifies its expression. This feature is important, because we can extract not only the noun but also the words which are developing it and therefor create a multi-word terms that are much more specific. Consider this example *"Atomic power plant, Coal power plant"*. We can now distinguish that one document was about atomic power plant and the other about coal power plant and therefore more precise. We will combine the entity detection, restrict it only for nouns and we will add direct constituents in the dependency tree to these nouns to create multi-word terminology extraction system. With this description we have to create specific patterns for this task. Since we are looking for specific noun and constituent and direct nodes, we will use Predicate-Argument model, with altered restriction on predicate. That means we will create rules, which will define that we are searching for nouns, which are subjects, object, adverbial or complement and if there are attributes dependent on these nouns we will extract them too. When we would have a list of these terms, we can count the frequency of them and we could use this frequency to determine the significance comparable with the terminology extraction without the linguistic processing. However we need to have also the overall frequency of the terms over all processed documents. With these numbers we can search for an algorithm that will count us the relevance of the term for the document.

4.3 Summary

We have analyzed many techniques for IE from the text and have found out that they can be solved by using specific patterns that will be applied on the dependency tree. To implement the IE we need to use these patterns and the extraction task will be transformed in finding a specific subtree in the dependency tree with restrictions on part of speech tags in the nodes and their constituents. The job is not yet finished, we have to transform these extracted information into machine readable form and store them in some suitable data store.

5. Describing and storing the information

In the previous chapters we have discussed what can we extract from the documents. What we need now is to find out how to store the entities, relations and terminology. Its not that simple because we need to create a model, a description of what extracted information means. In the first section we will try to find a proper language in that we will describe these information. With the language and the knowledge of extracted information we will create a describing model. We will finish this chapter with analysis and proposal of a proper data store.

5.1 How to describe informations

Lets assume we have extracted entities, relations and terminology with their frequency extracted from the document. Entity is a real life object, it can be person, thing, animal or phenomenon. Entity is in a relationship with another entity if a predicate exists in the sentence on which the entity is dependent. Lets imagine that we have hundreds of these entities with relations. We need to represent these entities and connections in machine readable form to allow the machine to manipulate with them. To manipulate with them means that the machine can store and search over them.

Before searching for the proper language to code the data we need to look at how the data are organized. Three options are worth considering5.1. The concept of traditional relational database is that we have some primary key and other fields are depending on it. Is this our case? Although it seems that we have relations between entities we cannot determine which entity should be a primary key and which entity should depend on it. Because the entity can be as an originator and also as a target of a relation. Its not the way how we should look at the data. Do the entities have some hierarchical structure? Again the answer is no. Lets take an XML document, we have a root node and tree structure with other dependent notes. We cant say which entity should be the root entity on the top of the hierarchy and the rest is somehow dependent on it. Graph representation is consisted by nodes related to other nodes, with no node having more importance over the others. If we could represent our entities as nodes, the edge between pair

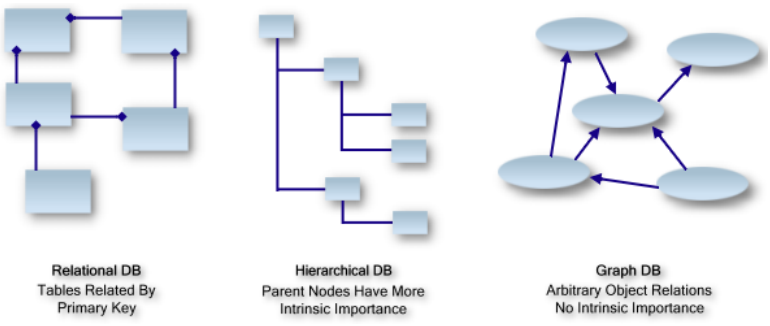


Figure 5.1: Data storage models. Taken from [40]

of nodes would determine that they are in some relation, if we could label these edges to specify what kind of relation it represents then we have found a proper way how to describe and store information extracted from the document. This graph is nothing else then already mentioned Knowledge graph. What we need to do now is to find how to describe this graph in machine readable form.

5.1.1 RDF definition

If the graph data model is the model designed for information extracted from documents, RDF is the language in which is coded. The Resource Description Framework(RDF) is a family of specifications designed by the World Wide Web Consortium (W3C). It is used as a general method for modeling various information as resources on the Internet, using a variety of syntax notation and data serialization format. RDF is designed to be easily read and interchanged by computer applications. It uses URIs (Uniform Resource Identifier) to identify resources. Resource in our case is an entity extracted from the document, relation, terminology expression. The URI consist of prefix and the resource name. For example `"http://datlowe.org/semjobKB/data/document/term#director"` has a prefix `"http://datlowe.org/semjobKB/data/document/term#"` and resource name *director*. In the same manner we will code other resources. A combination of a Resource, a Property and a Property value forms a *statement* known as subject, predicate and object *triple*. The subject denotes the resource, the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. The subject of an RDF statement is either a URI or a blank node, both of which denote resources. Resources indicated by blank nodes are anonymous resources. They are not directly identifiable from the RDF statement. The predicate is a URI which also indicates a resource, representing a relationship. The object is a URI, blank node or a Unicode string literal.

- `Resource` is anything that can have a URI such as `"http://datlowe.org/semjobKB/data/document/term#director"`
- `Property` is a Resource that has a name, such as `"http://datlowe.org/semjobKB/data/document/predicate#hasSkill"` or `"http://datlowe.org/semjobKB/data/document/predicate#coordinates"`
- `Property value` is a value of a Property (text, number) called *Literal* or it can be another Resource. Example `"http://datlowe.org/semjobKB/data/document/term#english"` or `"MarkO'Reily"`

A collection of RDF statements represents a labeled and directed graph. The RDF graph can be stored in any relational database but its rather stored in a native representation called *Triplestore*. RDF uses several serialization formats, most common is *RDF*
XML, N3, Turtle. We will use *Turtle* which is compact, human-friendly format. We will describe the database in a later section. We have described what is RDF, triple, resource, URI, Literal. We have to answer now the question, how we will code our extracted information.[20]

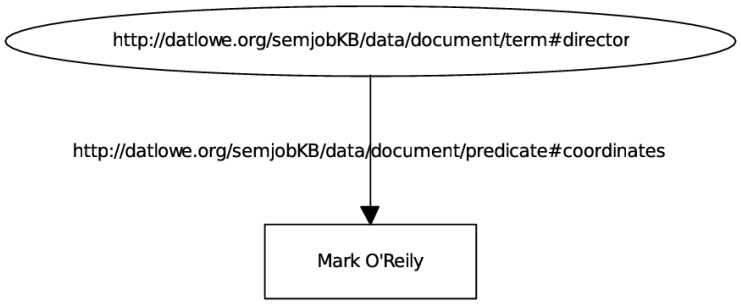


Figure 5.2: RDF triple example

5.1.2 RDF Schema definition

Is a set of classes with certain properties using the RDF extensible knowledge representation language, providing basic elements for the description of *ontologies*, otherwise called RDF vocabularies. It is used to structure RDF resources, to create a model and express relations between resources in the RDF document. Based on the RDF Schema we can have an overview over the knowledge that we are coding using RDF. As mentioned in the definition, ontology is a set of concepts within a domain, using a shared vocabulary to denote the types, properties and interrelationships of those concepts. For more details please look at `http://www.w3.org/TR/rdf-schema/`. The ontology based on information described in the previous subsection will be in detail described in the implementation part of the thesis.

5.1.3 How to code extracted information

We have a document from which we have extracted information. It is a set of triples (subject, predicate, object), set of terminology multi word phrases with the frequency in which they occur in the document. Each of the matched words have textual representation and lemma. Due to the linguistic inflection, a same object can have multiple textual representation, but all these words have a common lemma. To uniquely identify the entity (resource) we need to use lemma as a representation of the resource. URI will therefor have a prefix followed by a lemma of the resource. For multi word resource we need to replace the white space between the words with underscore character, `"http://datlowe.org/semjobKB/data/document/terminology#sale_department"`. URI are case sensitive and we need to convert every lemma to lower case. Every extracted part of the triple has to be transformed from its textual representation into the URI form by using its lemma. The frequency of a terminology is a positive number, we do not need to uniquely identify this number, therefore it can be coded as Literal.

5.2 Modeling our informations

The extracted entities can be grouped into 3 categories, terms as subject and object, because these two entities can appear on both positions in the triple and they are nouns. The second group is consisted from predicates, because predicate has to be a verb. Terminology has at least 1 noun as a root and dependent

adjectives that develop the noun. Some of the terminology can appear as a subject or object, however they contain different information for us and therefore should be separated into a unique group. This groups are common across all documents and each member of the groups can appear in multiple documents. As we have spoken, one graph is consisted of numerous triples. Each of the document has to have a unique graph that will contain extracted information. In the previous section we have described how to code individual entities but we need to find out how to code the whole document and lately the whole extracted domain.

5.2.1 Representing a subject, predicate, object

Each of the words has numerous textual representation with 1 lemma. By using its lemma we will create a unique URI for a specific resource. But we cant invoke linguistic processing each time we will need to get a lemma for a word and from that lemma the URI. We have to code it all at once. We will create one graph that will contain all textual representations of the subjects and objects. We will form a new triple where on subject position will be the subject or object URI, we will create property *prefix#hasTextualRepresentation* and the property value will be a literal containing the textual representation of the entity (subject, object) extracted from the document. Same will be for a lemma, that will store the precise textual representation of the lemma. In this way we can store all subject and objects at one place. We have not mentioned yet that each document will have its own unique graph. This approach much more simplifies the complexity of storing and searching over extracted data. We could store everything on on place, but logically, one document is one entity one resource containing smaller peace of information (triples, terminology) about the document. However we need to store all textual representations of the entities found across all document. Therefore we need to have one graph for each group that we have mentioned earlier. To recapitulate all above. There has to be one graph containing lemma and textual representation for the subject and objects and another one for predicates. When we would need to get a resource URI for a given word, we will look into this graph and we will search for a resource URI that has on a object position text that matches a given word. By that we can store in the document graph only URIs making the search easier. How to search over RDF graph will be explained in the next chapter.

5.2.2 Terminology representation

An extracted terminology is multi word phrase with frequency in which it appeared in the document. Textual representation and lemma should be treated in the same manner as with subject, predicate and object and so stored all at one place. The frequency is important information and has to remain in the document's graph. But we can also use this information and store the overall frequency of some terminology over all document together with the number of documents in that appeared. From this knowledge we can decide if some terminology is unique for some document or its common terminology that has small importance. Each document graph will have terminology URI on a subject position, then predicate *prefix#hasFrequency* and the literal with the frequency value. And at

one place we will store the all textual representations of terminologies and with their lemma, overall frequency, number of documents in which they are presented. This approach separates the common and unique information regarding a document and simplifies the knowledge representation and information redundancy in the graphs.

5.2.3 Document representation

Each of the extracted triple will have stored its textual representation in one graph and therefore we need to store only the entities' URIs in the document. One example can be: `"http://datlowe.org/semjobKB/data/document/term#director"` `"http://datlowe.org/semjobKB/data/document/predicate#speak"` `"http://datlowe.org/semjobKB/data/document/term#english"`. That information expresses that a director has a ability to speak English. To write RDF using Turtle syntax, each triple has to be on new line and has to end with a dot. The set of triples can then be stored in a database. In a same way we will add terminology to the document's graph by creating triples and adding them. The triple will contain the terminology URI, predicate *hasFrequency* and literal containing the frequency number. Example: `"http://datlowe.org/semjobKB/data/document/terminology#sale_department"` `"http://datlowe.org/semjobKB/ontology/document#hasFrequency"` "5". That means that the document contains 5 phrases with the lemma *sale department*. As you might noticed, we do not store which phrase exactly is in the document. We can add the textual representation in the document graph, however we do not need to which instance of the resource is in the document. We know that the resource, the entity is in the document and that is on what we are focusing. On the knowledge not on the instance of that knowledge. To retrieve a document from which a graph was created we need to add a triple which will contain the path to the document and also the path to the document in a plain text to display the text regardless the origin of the document. Also some statistical information as number of triples, unique triples, predicates, terms, sentence count would be useful. The triples and the terminology extracted and coded into the RDF using Turtle syntax then can be stored in the document and we have not loss any information in this phase. We do not have to forget on blank nodes. Sometimes we can only extract a part of a triple, for example there is a missing object. We do not want to loose at least this partial information therefore we will use a special resource *prefix#blank* URI to make it clear that the triple is incomplete. Everything what was extracted will be stored in the database and overall knowledge as total frequency and textual representation over the documents will be extended.

5.3 Searching for a proper database

The graphs created from the knowledge acquired from the documents has to be persisted in the database to be able to search over them. The triples can be stored in the relational database however the database is not designed for these kind of data. A (triplestore) is a purpose-build database for the storage and retrieval of triples. Like a relational database, one stores data and retrieves it vie query language. A triplestore is optimized for the storage and retrieval of triples.

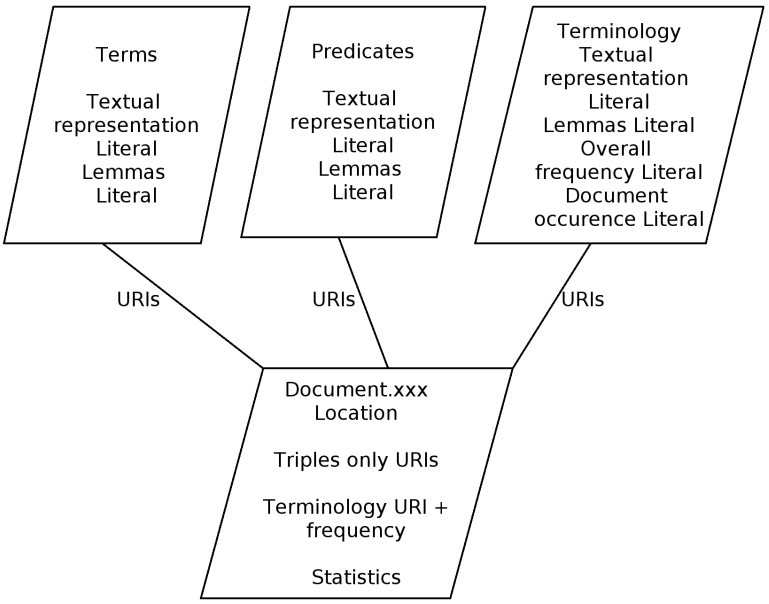


Figure 5.3: Model overview

Moreover triples can be imported and exported using RDF. There are several implementations of triplestores:

- **FUSEKI** is a HTTP server that allows SPARQL query language to PUT/-POST/DELETE RDF triples. It runs only in memory, those making it unsuitable for permanent persistence of graphs.
- **Oracle** is an open, standards-based, scalable, secure, reliable and performant RDF management platform. Based on a graph data model, RDF data (triples) are persisted, indexed and queried, like other object-relational data types.[41]
- **OpenRDF Sesame** is a de-facto standard framework for processing RDF data. This includes parsers, storage solutions, reasoning and querying, using the SPARQL query language. It offers a flexible and easy to use Java API that can be connected to all leading RDF storage solutions.[42]
- **OpenLink Virtuoso** is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system.[43]

Our requirements are pretty simple, we need to permanently persist graphs in the database, update them when using different search strategies and to search over them. The database has to be able to store hundreds of graphs. The performance is another important aspect. Hundreds of graphs will consume large space and therefore in-memory database is not and option, because treex consumes a large amount of memory when running, those we need to leave a space for it. The database has to also provide indexing for faster query searching. Oracle and Virtuoso are suitable for the purpose. From previous experience and Virtuoso and Jena Java API the Virtuoso is a perfect candidate. Virtuoso solution contains

Virtuoso Conductor that is a Web Based Database Administration User Interface trough that we can access to the graph, use SPARQL query language (will be explained in the next chapter). *Virtuoso Jena API* is a API to access database directly from Java project, those making it simple to interact with the database, more details in implementation section.

5.4 Summary

We have elaborated the possible representations of the extracted information. We found out that the RDF is designed for describing and modeling information and its suitable for our task. We have discussed how to model our extracted information and how we will create and store graphs. In the end of the chapter we have searched and select appropriate database engine. The only remaining research has to be done upon designing a search engine over the graphs.

6. Search Engine

Lets assume that we stored knowledge in graphs and these graphs are stored in the database. We have to propose what kind of search types are interesting for us. This topic will be discussed first, then we will explain the SPARQL query language that is used in Virtuoso.

6.1 Available informations

Document graph contains full or partial triples, partial means that a triple lacks one of its part. Triples are consisted of resources' URIs, on a subject and object position is a term, predicate interconnects these two parts. Additional informations as textual representations and lemma are stored in two separate graphs for terms and predicates. In these graphs on a subject position is the resource URI of the subject and object is a literal containing the string textual representation or the lemma. One unique textual representation is stored in exactly triple. The terminology words are stored also in a document graph. One triple per each terminology, subject has the resource URI and object is a literal that has frequency number of the occurrences of the terminology in the document. Textual representations and lemma are stored on one place in a same way as for terms and predicates. Additionally we will store the overall frequency of terminologies among all documents and a number of documents in which that particular terminology was found at least once. That will allow us to identify rare terminology, unique for a certain set of documents and also we will be able to sort search results based on the number of occurrences of the terminology. For a research purposes it would be wise to store in each document a small statistics as a number of matched triples, count of unique terms and predicates. Because terms and predicates can occur in multiple triples it will be expensive on computational resources to calculate these numbers every time we will need them, therefore we will have them stored in document graphs. Also we would definitely need for statistical purposes the number of sentences that document has, has means that the linguistic tool recognized the set of words as sentence. The document has to have also the path to the original document together with the path to the extracted plain text that was used in the linguistic processing.

6.2 Search specifications and possible solutions

Fundamental search requirement is to find a documents matching specific information. The specific information is a full or partial triple and terminology. Since the document contains only URIs we cant ask user to write URIs in the search query. All search has to be done by using words. This implies that we need to search for a resource containing a this word in textual representations or in lemma. Once we get the resource URI we can continue to search over the documents. Since the terminology contains frequency information we will divide the search specification into 2 parts. For triples(subject, predicate, object) and for terminology.

6.2.1 Search based on triples match

Traditional search is done by writing a keyword or keywords and then a search engine is trying to find the keyword in the text. It goes trough the document and matches every occurrence of the given input regardless of the importance that the keyword carries in the sentence. On the other hand we extract only words that has a specific meaning or more precisely a specific function in the sentence. Since we have extracted subject, predicate and object from the document and created a triple from that, we can use this as a specification for our search. We need to design system that will allow to search over the documents based on the keyword on the subject, predicate and object position. We will create a 3 fields, one for subject, one for predicate and one for object. The keywords inserted into the field will be matched against the terms graph for subject and objects and predicate against the predicate graph. We will try to find resource that has textual representation or lemma matching with the given keyword. If found the URI will be taken from that resource. We will create a triple from these matched resources and that triple will be our search query against all documents graphs. If the resource would not be found, we have to give a warning that not for all 3 field the resources were found. However if at least one resources has been found we will try to find matching documents. For example predicate was not found, but subject and object resource were. We will search for a graphs having triple that has subject and object matching with our. The result document set will be returned. But the graph itself would not be a proper result to display. Thats why we will need to post-process the graph and extract a path to the original file, the text extracted from the original document and the list of triples and terminologies. So the user would be able to open the document or check the text if the found document suits his search intentions. Lets look at the figure6.1 to see the process.

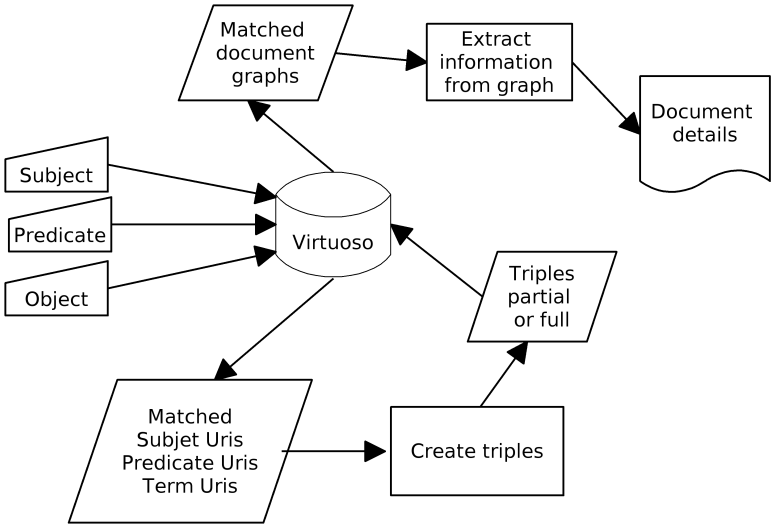


Figure 6.1: Search based on triples

6.2.2 Search based on terminology match

Terminology has an extra information, the frequency, overall frequency and number of documents in where it is presented. We need to use this information

to distinguish between common and rare term. There are different algorithms that can calculate the relevancy of a given terminology based on its frequency in document, on number of other documents containing this term over all documents in the database. These algorithms are designed for terminologies that have no linguistic preprocessing. That means the algorithms do not know if a terminology also carries a specific information important for that sentence and also for the document. We have to combine the common terminology algorithm with our linguistic extraction approach. In any case we need to return all graphs containing this terminology. The search flow will be the same as described in the section with triples. We will take a phrase, look into the terminology graph to find URIs containing the given input and then search for the graph which contains given URIs. Since the terminology can by multi words phrase we need to search for any textual representations or lemmas containing the whole sequence of input words or containing these words. But what we can add to this basic search based on matching words is to calculate the relevance of the matched resources among all documents.

6.2.3 Relevance of the terminology

To calculate the relevance of a terminology we need to take into account the overall number of documents, the frequency of terms in the document, number of words in the document and number of documents containing the terminology. We can use *TF-IDF*[44] (Term Frequency - Inverse document frequency) algorithm to calculate the relevance. The TF-IDF has two parts. TF calculates the terminology weight based on the terminology frequency n_k divided by the number of words $\sum_i n_i$ in the document $tf_t = \frac{n_k}{\sum_i n_i}$. IDF equation is calculated from *logarithm* of total number of documents N divided by the number of documents df_t containing given terminology $idf_t = log\frac{N}{df_t}$. The relevance is calculated by multiplication TF and IDF values $tf - idf_{t,d} = tf_{t,d} * idf_t$. The result value is high when terminology is frequent in the document and there is only a few document containing the terminology. If the terminology is common for many documents or if is less frequent in the document the TF-IDF value is small. We need to take into account that the total number of words, since we are using linguistic processing is pointless for us, because we are only extracting terminology carrying some knowledge in the sentence. However we can alter this calculation by swapping the total number of words by total number of terminologies in the document. Based on the observation a terminology that was marked by linguistic processing as important in one sentence does not need to be marked as important in other sentence, the NLP always might lower terminologies frequency compare to non NLP terminology extraction and the overall frequency of all terminologies can only be the same or lower then without NLP. Therefore compare to the original TF we will not only lower the nominator and also the denominator and the sense of the TF will remain unchanged. Since all the data values are persisted in the terminology graph we can calculate the TF-IDF for each terminology in the document and display them to the user.

6.2.4 Non document search specification

There are many other search possibilities over the data. We do not need to search only for documents. Having the triples stored in the document, another important information might be what kind of triples do we have, more precisely which terms or predicates are connected with others. For example we are interested in finding all connections for some resource. We will write the resource as string on the subject position and we will search for all edges (predicates) going out from that node to another node as object. We can call it as finding the remaining parts of the triple. Once we have specified the full triple we can search for the document having this kind of triple. This can be done for all positions and also we can fill 2 out of 3 parts and search for the last one. The user would then see the connections that exists in the database and specify in what he is interested in see figure6.2. For a specific resource, does not matter if it is term or terminology, would be good to show which textual representation does it have. For example we would like to find a specific resource URI and all textual representations together with lemma that it has.

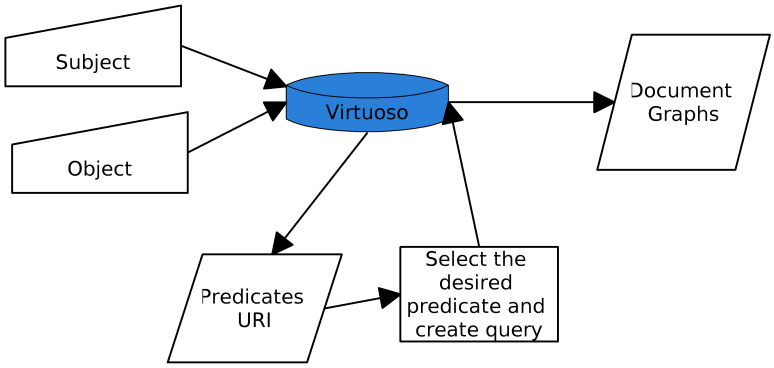


Figure 6.2: Complete the triple and search for documents

6.3 SPARQL Definition

SPARQL stands for SPARQL Protocol and RDF Query Language is an RDF query language for semantic databases able to retrieve and manipulate data stored in RDF format. SPARQL allows query to be consisted of triple patterns, conjunctions, disjunctions, and optional patterns. Its structure is similar to SQL query language, however SPARQL allows users to write queries against data that can loosely be called *"key-value"* data, more specifically it is data that follows the RDF specification of the W3C. The entire database is thus a set of *"subject-predicate-object"* triples. More information can be found here[45]. Example query:

SELECT ?graph ?predicate WHERE { GRAPH ?graph {<`http://datlowe.org/semjobKB/data/document/term#director`*>?predicate "english"}}*

We would like to search for a graph and predicate which contains triple where on a subject position is a resource *director* and on object position is Literal string

ënglish:. The syntax of the SPARQL is pretty straightforward as you can see. All semantic database allows to use SPARQL and therefore we will create our search queries based on the SPARQL syntax.

6.4 Conclusion

For every result of a search query that will return documents or resources, we need to display all knowledge that is related to that entity and it is stored in the database. For document it means, its statistics, text, triples terminology with their frequency and their TF-IDF index. For resource its the textual representation, lemma. All this requirements could not be done without Graphic user interface. We have to design one that will enable user to use all functionalities which we have described in this theoretical part of the thesis. To summarize what we have:

- Documents as testing data
- Linguistic tool Treex
- Knowledge of what is important in the sentence and how to extract this knowledge
- Virtuoso semantic database that will allow us to store the extracted information
- SPARQL query language and search specification

Lets design a system that will implement all these things and knowledge into one powerful software.

7. Implementation

This chapter opens the implementation part of this thesis. The goal of this thesis is to design a software that will experimentally test proposed solutions of the tasks described in the theoretical section with the documents received from CEZ, a.s. company. In the beginning of this chapter we will discuss requirements on the software, then we will focus on architecture and its composition. The following chapters of the implementation part describe each module in greater detail and enlist problems that we have encountered.

7.1 Requirements

The software has to be able to extract informations from documents of various types (Microsoft Word, Excel, Power-point, PDF). This task includes extraction of plain text from these documents, invoke linguistic processing, process the linguistic results, create and apply search rules on them. The extracted knowledge has to be converted into graph representation and stored in the Virtuoso database. Part of the application is GUI that will allow a user to interact with the rest of the application. These are the general requirements for the application based on the goals of this thesis. To summarize the requirements:

- Extract plain text from documents
- Process extracted text with the linguistic tool *Treex*
- Process the *Treex* results
- Provide an environment to create search rules for IE
- Apply and extract informations
- Create a graph representation of the IE results
- Store the information in Virtuoso database
- Provide user interface to search and retrieve data from Virtuoso
- Gather statistical informations about the informations extracted from the documents

Although this is an experimental application, we should consider also future usability and extensibility of this application that will lead someday into a real-life implementation. That implies use of a central database storage and a machine that will provide enough computational power for linguistic processing tool *Treex*. These requirements are met in a client-server model, thin-client implementation. Application therefore will be deployed on the server as a web application, database and *Treex* however do not need be deployed on the same server, to reduce a server workload we have to design the application that will allow to have them on a remote machine. Clients will work with the application trough web browser. Application will be developed in *Java* programming language and will be using *Spring framework*.

7.2 Architecture

We have described software requirements in the previous section and proposed a model of our solution. We have to find a suitable architecture for our application. The task can be decomposed into smaller ones, allowing us to create modules that will implement individual requirements. However the requirement can change in time, so we need to design these modules in a way that will minimize impact on implementation of other modules. Since user will interact with the application trough a web browser using thin client approach the MVC (Model-View-Controller[46]) design pattern will be the most appropriate. *Controller* will react on user events from the *View* component and will send requests to the *Model*. Once he will get the notifications (results) from the *Model* it will update the *View*. See figure 7.1.

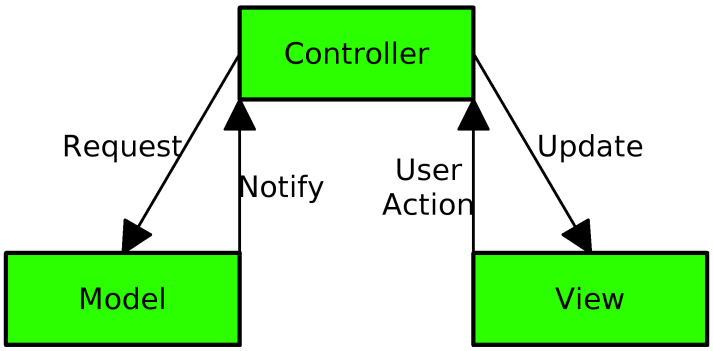


Figure 7.1: MVC design pattern

If we consider the requirements and the definition of the MVC pattern, we have to divide modules into *Model*, *Controller* and *View* components. *Model* contains domain logic and therefore all modules that will be serving this purpose will be added there. View will be a web browser UI, Controller can be any class that will interconnect all modules. The modules have to implement specific interfaces to provide uniform access from *Controller*. The MVC pattern does not describes the interaction between domain and data layer and therefore we will implement our own solution.

7.3 Architecture in details

The application architecture is shown on the figure7.2. *View* and *Controller* will be implemented in a same model, they do not need to be in a separated module explicitly. There will be 5 modules in the *Model* component. *Treex* and Virtuoso database can be deployed on different server and we mark them in the architecture as remote entities.

7.3.1 Model

Each of the module implement part of the requirements and represent the application's domain logic. To provide extensibility, each of the modules will be implementing interfaces. Any future module that will implement any of the interface can replace the module. it will minimize the chance of affecting implementation of other modules. Modules that are part of the *Model* component:

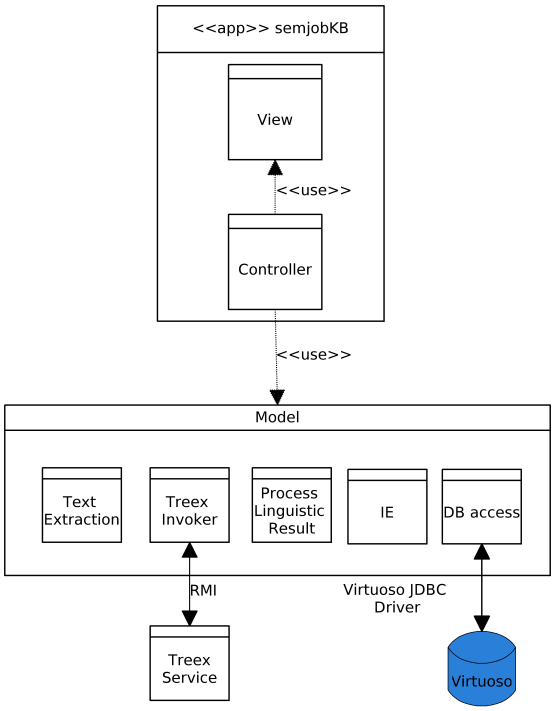


Figure 7.2: Software architecture

- *Text Extraction* - Will receive any of the document provided from CEZ a.s., will extract a plain text from it, will store it in a file and will return reference to that file. The supported document types will be Microsoft Word, PowerPoint, PDF. During the extraction process the structure of the text must be intact. To reach this goal some post processing will be needed.

- *Linguistic processing* - Will receive a plain text file and will invoke a linguistic text processing. In this case it will invoke *Treex*. However this module could be switched by any of linguistic processing tools, for example *Stanford CoreNLP* and others. To support extensibility and load balancing, it will invoke the linguistic processing remotely. To support this feature, we have to invoke this service using Spring RMI.

- *Process linguistic result* - Treex writes results in *CoNNL*[31] format. The modules will get a file containing the results of linguistic processing in this format. It will parse the result and will create an internal representation of the document, that will contain all linguistic informations about the document such as dependency trees of the sentences and for each word its morphological tags, lemma, original text etc. It will return object that will contain the internal document representation to allow IE based on specific patterns.

- *IE* - This module will receive internal document representation that will be created in the previous module and search configuration file. Search configuration file will contain information about the search patterns. The module will process the document based on the search configuration and will extract triples and terminology expressions with their frequencies. The

module will return an object that will contain set of extracted triples and set of terminologies.

- *DB access* - This module will provide access to the virtuoso database, or any other semantic database that will implement specific interface. This module will create a graph from information extracted from a document and store them in the database. On top of that, the module will enable to search for a document based on parts or full triples and terminologies. Besides document search, it will allow to search for triples and terminologies. It will return any information that is stored in the database based on the type of a request. The module will use Virtuoso JDBC driver to connect remotely to the Virtuoso database.

7.3.2 View and Controller

View component will allow user to interact with the application. The application needs to be accessible via Web browser, since domain logic will be written in *Java*, use of *Servlet* is almost obligation. To simplify interaction between the UI and the rest of the application we should look for a Java framework that will allow us to create UI and easily embedded it into our application. One solution might be a GWT (Google Web Toolkit) which is development toolkit for building and optimizing complex browser-based applications. Another solution is Vaadin that is ava framework for building modern web applications. Both options are suitable, however *Vaadin* is relatively new, it focuses mainly on server-side architecture and uses GWT for rendering the resulting web page and it would be nice to learn something new. As mentioned earlier Controller can be any class that will interconnect modules, processes user requests and pass them to the domain logic. In the opposite direction it will receive updates from the domain logic and will fill the UI with a new data.

8. Implementation of IE

In this chapter we will describe the implementation of IE in the application. The IE process has a form of a pipeline, where on the input of the first layers is a document and on the output of the last layers are extracted informations. Each layer's process is implemented in exactly one module. The pipeline workflow is physically implemented in the *Controller*, the figure8.1 represents the logical workflow.

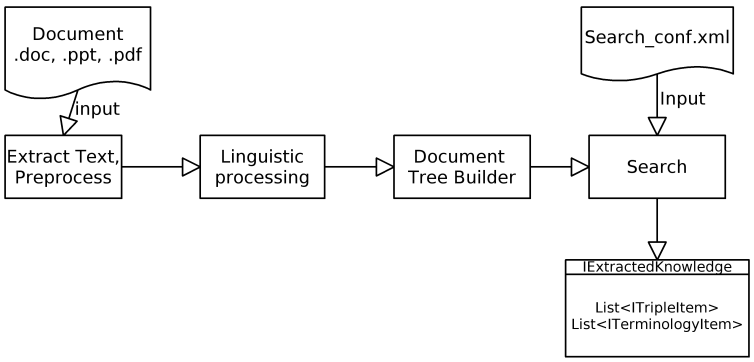


Figure 8.1: Implementation of IE pipeline

8.1 Plain Text Extraction

The linguistic tool *Treex* can process only documents in a plain text format. We received documents in .doc, .ppt and .pdf format. To be able to process them we need to extract plain text these documents to be able to pass them for the linguistic processing. The text extraction process cannot change the structure of the text to have relevant input for the NLP. This might be difficult, since the documents contain tables, enumerations, references and inner documents. PowerPoint presentation from the structure perspective are most complex. Slides are consisted of objects with text and pictures, object can have more nested textual objects inside. However we need to find a way how to extract the text and then we will deal with the structure. Apache POI is a *Java* API that provides methods to extract the text from Microsoft Documents. Word documents (HWPF+XWPF) and PowerPoint presentations (HSLF+XSLF). They accept Microsoft Documents created in version before 2003 and also after. Apache PDFBox is *Java* library for working with PDF documents enabling to extract text from them. We will use these Open-source libraries.

8.1.1 Implementation

Figure8.2 shows the class diagram of this module. *ExtractTextService* implements *IExtractTextService* interface with the method *extractText(File): String* that takes document file and returns String containing a plain text. Based on the type of the file, the corresponding class using the *Apache POI* or *Apache PDFBox* libraries will be called to extract the text form the documents. When the extraction phase

is over, the text is send for post processing to reorder the text to match the original document's structure. The text is then stored into the subfolder where the original document is stored to skip processing of the same file again if needed.

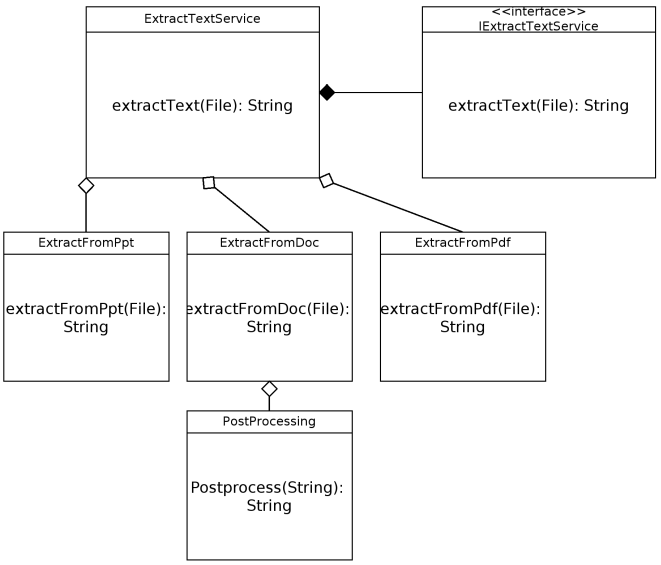


Figure 8.2: Text Extraction Diagram

8.1.2 Observation

The process extracts all text from the document without any losses, but the structure of the documents sometimes differs in the plain texts. PDF files are extracted without any structural changes. The problem comes with Word and PowerPoing documents. The extracted texts contain additional empty lines. They do not have impact on Treex performance, however if we would like to edit these documents, it does not look right, therefore we had removed these empty lines and restored the suitable structure of the documents. In PowerPoint presentation the structure of the text is changed. The documents in the Experts' Profiles have the same structure in the original document. But the plain text files differs and moreover the structure differs even between the plain text files. The *Apache PDFBox* library creates its own internal structure of the document, it processes each object one by one and appends the text on the output. The different result in each plain text suggest that even if the PowerPoint presentation looks the same, the nested objects do not have the parent objects set identically in each of the presentation causing to have different order of processing. Any attempts for reordering failed due to this random behavior in the results. In the Word documents, the text extraction of paragraphs and texts is working well. During the processing of tables, the hidden table names and columns are shown in the resulting text. We cannot determine if the word is a table name or if it is a part of the text and therefore we have to admit that they will appear in the result. The problem cannot be solved by some general solution. Without broad testing of each document groups and optimizing the extraction process for each particular group individually we cannot get close to the correct results.

8.2 Linguistic processing

In the theoretical part of this thesis we have described the NLP tool Treex in details. In this section we will show how it is implemented in the application. Treex runs only on Linux based systems. The Treex takes as input a plain text file, processes it and writes the result in ConLLX format into an output file. Every time the Treex is invoked, it loads all required modules into operation memory and takes around 3GB of memory. The memory usage is significantly determined by the chosen parser. We could use smaller models, however it will have impact on the precision of processed text. We have chosen *MSTParser*, that has decent precision and memory requirements. Besides high memory usage during processing, it takes almost all CPU time. This causes the Operation system to lag. Considering this we need to allow to run Treex on different server then the rest of the application. The simplest solution will be by invoking Treex service via *Spring RMI*. This module therefore will be split into a client side and a server side.

8.2.1 Implementation

The NLP module is shown on the figure8.3. The client is implemented in *LinguisticService* and offers the application to call NLP in method *linguisticPocessing*. The method accepts plain textual file, it reads the text and calls remote method *remoteNlp* and waits for the result to come. Once the result will be available, it will store the file in a subfolder to avoid processing of the same file later again. The server side contains class *RemoteNlp* that implements *IRemoteNlp* interface and provides method *remoteNlp* which receives a text, invokes Treex and reads the result from the output file and returns it back to the client. Invocation of Treex is done directly from the code by calling *Runtime.exec()* method.

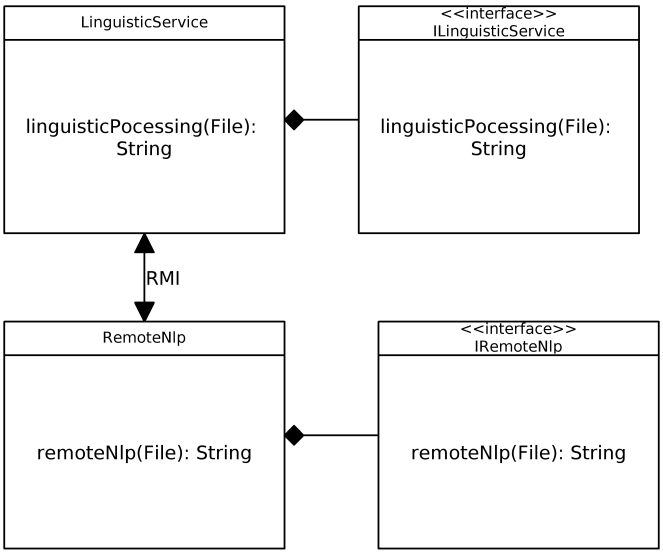


Figure 8.3: NLP Diagram

8.3 Observation

Treex does not allow to have all required modules preloaded in the operation memory. Every time new NLP request comes, it has to load all models from disk to the memory which takes about 40 seconds. After the loading phase is over, the Treex performance is then determined by the size of the text. It takes about 5-10 seconds for 1 Word page based on the text density. In most cases the duration of loading phase exceeds the processing time. For future business use it will be appropriate to have all models preloaded in the memory. This will allow processing of hundreds of texts, otherwise the NLP will be a performance bottleneck of the application.

8.4 Document Tree Structure

The result of the NLP is stored in a textual file. To make searching easier it would be better to load the result into application object and then apply the patterns over its internal structure. We can call it as Data Access Object(DAO). The purpose of this module is to create this DAO object and fill it with the data of the NLP process.

8.4.1 Implementation

Document has a hierarchical tree structure. Root node is the document, its direct childes are paragraphs, paragraphs have sentences and the sentences has words as leafs. Figure 8.4.

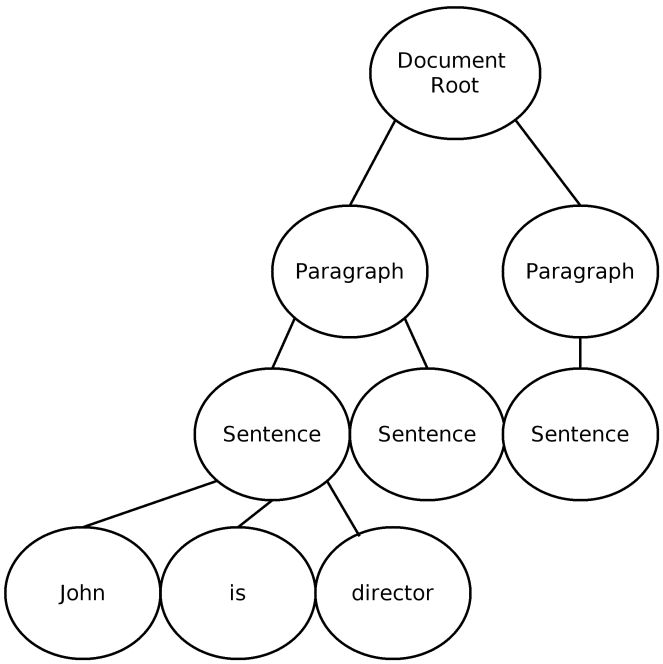


Figure 8.4: Document Tree Structure

All the information about the ConLLX Treex output format together with the example3.1 can be found in the theoretical section. The idea is, that all linguistic

informations are connected only with words. Paragraphs and sentences are used in the internal structure as a reference to ordering of the original document. Information related to the words are:

- Sentence has a list of words, index of a word in the list determines its original position in the sentence
- Textual representation of the word
- Word lemma
- Morphological informations such as word type (noun, verb, ...), genus, number, fall, person
- Constituent (subject, object, predicate, ...)
- Index of a parent word on which the word is dependent. Used to create a virtual dependency tree of a sentence
- Information if the word is on the left or on the right side of the parent node

The module receives a result, it creates a document root, adds new paragraph on the root, new sentence on the paragraph and starts processing the file line by line. Each non empty line represents NLP result of one word in the sentence. All informations are extracted from the line and filled the word leaf object with the data. If a line is empty it means end of the sentence. The module then creates a new sentence object and adds to the currently active paragraph. When we reach the end of the document, our internal document structure contains all data from NLP of the document and returns it.

8.5 IE implementation

The purpose of this module is to extract informations from the document, based on the search configuration file and the NLP data result. The output of this process are list of extracted triples together with terminologies and their frequencies. Search is invoked from *SearchService* class by method *extractBase(IDocument document, ISearchRules))*.

8.5.1 Search rules and search process

Search is based on finding subtree in the sentence dependency tree that fulfills constraints on word nodes. In the theoretical section we have explained for what we are searching in the document, entity detection and relation extraction. We are searching for triples called subject-predicate-object. A predicate denotes the relation between subject and object. A subject is an originator, in other words it denotes about what a sentence is. An object is the target of a sentence. We have 2 possibilities on how to create and apply patterns. We can create a rule that will contains all tree parts. Each part of the rule (subject, predicate and object part) can have a various types of constraints, to cover all possible patterns we would need to combined each of them resulting in dozens of patterns that

will be applied on each sentence. There will be patterns that will differ only in one constraint causing a lot of redundancy in searching. Better approach is to create and apply search rules for each part of the triples individually and combine them on a common predicate. Terminology extraction consists from one part in contrast with triples. We are searching for any noun that with significant meaning determined by the constituent in the sentence. The search rules are:

- **Subject** - Has to be dependent on the predicate directly or its parent node is dependent on the predicate and the parent node is an auxiliary node. Only *Subject* constituent, word type noun.
- **Predicate** - Any predicate constituent in the sentence with an auxiliary verb that develops the predicate.
- **Object** - Has to be dependent on the predicate directly or its parent node is dependent on the predicate and the parent node is an auxiliary node. Allowed constituents are *Object, Complement, Adverbial* and word type is noun.
- **Terminology** - Any noun that has constituent *Subjects, Object, Adverbial or Complement*. Any attribute directly dependent on this node is appended in the result

With the search rules being set up, advance to the search itself. For triples:

1. For each sentence repeat:
   (a) Apply rules and find all predicates, subjects and objects
   (b) Filter the duplicate matches
   (c) For each subject find which predicate in the sentence is dependent and create a pair with the predicate
   (d) For each object find which predicate in the sentence is dependent and create a pair with the predicate
   (e) For each subject and object pair create a triple if they are dependent on the same predicate
   (f) Add the triples found in the sentence into the list of document's triples
2. Return result

To avoid loosing informations from the sentence when the object is missing, we will use the partial result as double with an empty object. Terminology search algorithm is similar to the triples search:

1. For each sentence, apply the rules
   (a) If a match was found, try to find any child nodes that are developing the match and are allowed by the rules.
   (b) Reorder the words based on the position in the sentence and remove duplicate matches
2. Group the terminology matches based on their lemma

3. For each group create the result match that has common lemma, unique textual representations and size of the group as frequency of the term in the document.

Remove duplicate matches in the search algorithm means, that if on rule is also a subpart of another, then if the search finds the larger one, than it had to found the subpart and we have the duplicate matches. After all matches has been found, the algorithm goes trough the list of matches and removes the duplicate sub matches. Both, list of triples and list of terminologies are added into *ExtractedKnowledge* object and passed back to the *Controller* as a result.

8.5.2 Writing search configuration

The search configurations are written in XML document. When IE search module is invoked configuration is loaded into memory and it extract the rules for subject, predicate, object part of the triples and for terminologies. The default configuration is shown in the Appendix C12.2. We will explain the parts of the search configuration:

- *ruleset* - contains set of rules for each search part. The *type* attribute denotes to which set it belongs. Possible values are *predicate, subject, object or terminology.* Ruleset node contains individual rules.
- *rule* - represents one particular rule and it contains one or more nodes.
- *node* - It represents a restriction given on the node (word) in the sentence dependency tree. Each child tag creates a restriction on constituent, parent word constituent and word type of the node.
- *constituent* - restriction on the constituent of the node in uppercase
- *vassal* - marks the node as a child of another node. Values true or false
- *parentType* - restriction on parent node constituent in uppercase or null if the node is root node of the rule
- *wordType* - restriction on the word type of the constituent in uppercase null if the restriction is not needed

User can alter or create a new rules and apply them on the IE from the documents.

9. Database Implementation

We will start with describing the ontology model used for creating hierarchical structure of our domain. Then we will explain how to connect and work with the Virtuoso database. Then we will clarify how the extracted information will be converted into the graph and stored in the database.

9.1 Ontology

This section is a recapitulation of the theoretical section Modeling the information5.2. Informations are divided into classes:

- *Terms* - Contains information about subjects and objects, its URIs, lemmas and textual representations
- *Predicates* - Contains information about predicates, its URIs, lemmas and textual representations
- *Terminologies* - Contains information about terminologies, its URIs, lemmas, textual representations, overall frequencies, frequencies, number of documents in that occur, TF-IDF
- *Document* - Contains document name, path to the original document, sentences, terms, predicates, terminologies count, triples and terminologies

The ontology is shown in the Appendix B12.2. When we were designing the ontology, we were focusing on how to hierarchically structure informations. What is relevant for the document and what is common for all documents. Resource (term, predicate, terminology) instance is relevant only for a document and it denotes that the document contains that kind of information. The textual representation is not relevant for it because it denotes only instance of a particular resource object. Therefore it does not need to be in the document's graph and we can group the textual representations in one common graph. We have created graphs that has well known URI's, textual representations and lemma of the resources. Using URI's prefix *semjobKB:*`http://datlowe.org/semjobKB/data/document/` the graphs are:

- semjobKB:term# for subjects and objects, graph contains individual textual representations of the resources and their lemmas
- semjobKB:predicate# for predicates, graph contains individual textual representations of the resources and its lemmas
- semjobKB:Terminology# for terminologies, graph contains individual textual representations of the resources, its lemmas, overall frequencies, number of documents in that occur

To optimize searching based on the textual representation of some resources, we do not need to go trough all documents, all we need is to look into these graphs, get the resource URIs having the same textual representations and search for documents based these URIs. This ontology allows us to efficiently store the informations and provide fast searching services.

9.2 Working with Virtuoso database

To work with the Virtuoso database from the application we have decided to look for a *JAVA API* that will provide us with methods to connect, update, store and retrieve data. That functionality is available in the open source Semantic framework called Jena. *Jena's* subproject called *virt-jena* `https://github.com/srdc/virt-jena` is especially designed to work with Virtuoso database and therefore we are using this library in the application. All it needs is to set the database url location, user login and password that are injected into the application in *config.properties* file. The library enables to create Virtuoso graphs, fill them with triples and store them. The *virt-jena* does not have any tutorials only *Javadoc* documentation and few example test that are part of the project.

9.3 Converting and storing extracted informations into a graph

The extracted informations that has been described in the previous chapter needs to be converted into the graph based on the ontology. We have list of triples, and list of terminologies with their frequencies. The process of converting informations into the graph:

1. Check if the document is exists in the database, if yes then:
   (a) For each terminology URIs in the document subtracts the overall frequency by the frequency of the terminology and decrease by one the number of documents in which the terminology occurs
   (b) Remove the graph from the database and create an empty one
2. else:
   (a) Create an empty graph
3. For each terminology:
   (a) Escape all characters in the lemma that are not allowed in URI and create URI from the term prefix semjobKB:Terminology# and this escaped lemma.
   (b) Create new triple for terminology, predicate semjobKB:hasStringRepresentation and literal having the textual representation of the terminology
   (c) Create new triple for terminology, predicate semjobKB:hasLemma and literal having the lemma
   (d) Crate or update the terminology overall frequency and number of occurrences
   (e) Add the triples into the terminology graph
   (f) Create triple from terminology resource URI, has:Frequency predicate and literal filled with the frequency
   (g) Add it into the document graph

4. For each triple:
   (a) Extract the subject and object from the triple
   (b) Escape all characters in the lemma that are not allowed in URI and create URI from the term prefix semjobKB:term# and this escaped lemma.
   (c) Create new triple for subject and object, predicate semjobKB:hasStringRepresentation and literal having the textual representation of the subject or object
   (d) Create new triple for subject and object, predicate semjobKB:hasLemma and literal having the lemma
   (e) Add the triples into the term graph
   (f) Make the same for predicate and add the textual representation triples into the predicate graph
   (g) Create triple from subject, predicate, object resource URIs and add it into the document graph
5. Create triples containing the sentences, terminology, terms, predicates count, language, document name, document path and store them into the graph.
6. Method graph.close() commits the changes made and uploads the graph into the database

This algorithm ensures that when we will process the same document again, maybe with different search strategies we wont loose terminology overall frequencies and the number of occurrences. The predicates used to create triples about document details and statistics are in the ontology and can be found in the Appendix.

9.4 Implementation

Implementation is done in module *dbVirtuoso* in a class DatabaseService that implements *IDatabaseService* and provides method *(IDocumentDetail storeDocument(IExtractedKnowledge extractedKnowledge, File documentFile, ELanguage language))*. The parameters are: extracted knowledge, file containing the original document and the language of the document. The language has default value set for the Czech language, but we were counting with other languages that might appear in the documents later on. The method *storeDocument* after successful creation and saving the graph returns the graph details to be shown to a user in the application GUI. We do not provide the functionality to remove the document from the database.

10. Search Implementation

In this chapter we will focus on how the search is implemented and used in the application. First we will expand the search requirements proposed in the theoretical part of this thesis. We will follow with the algorithms used to search for a stored informations.

10.1 Search requirements

The general requirements on search engines are to find document based on triples and terminologies that it contains. But we can provides much more richer search possibilities then only document search:

- **Search for triples** - by writing only subject, predicate, object or any combination of these parts we would like to get which triple in the database contain these parts. We can call it as specifying a search query for document search. User does not need to have the knowledge of the stored triples and therefore do not need to try and guess the full triple for document search
- **Search for resource** - this will allow to search for nay resource having particular textual representation or lemma. Its a test on the database, if it contains document with that kind of knowledge. This will not result into the document search, but only in the terminology and triple parts search
- **Document search based on a triple** - Search based on full triple or any subpart
- **Document search based on a terminology** - Search based on the terminology, the results needs to be sorted by the TF-IDF to provide more relevant search results first
- **Document search based on a triples, specified by a terminology** - Since parts of the triples have mostly one word, we need to provide a user a way how to specify more his search query. Example triple: *Smith works department*, department is a relatively general meaning. What we are trying to accomplish is that user will add *sales department* in the terminology search field. We will then trigger search for documents having this triple and this terminology in its knowledge graph.

As we have mentioned earlier, if the database contains a specific resource, it also contains its lemma and textual representations, therefore users do not have to write the URIs of the resources. The search will be based on the textual representation and lemmas.

10.2 Implementation

The search methods are available in the module *dbVirtuoso* in the class *DatabaseService* that implements *IDatabaseService* interface, see figure10.1.

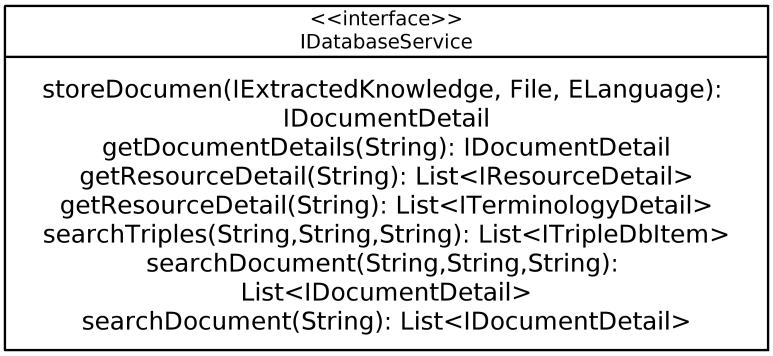


Figure 10.1: Database Service Interface

Method *storeDocument* which is part of the interface is described in the previous chapter and it transforms the extracted informations into graph and stores it in the database.

10.2.1 Search for triples

The search is provided by method *List¡ITripleDbItem¿ searchTriples(String subject, String predicate, String object)* and it works in two steps:

1. Determine which parts of the triple are specified
2. For each part find the resource URIs that has textual representation or lemma same as input text
3. If one part is specified then
   (a) For each resource URI
   (b) Replace the ?subject, ?object, ?predicate string with the resource URI in the database query template *SELECT DISTINCT * WHERE GRAPH ?graph ?subject ?predicate ?object*
   (c) Process the result and add triples into the resulting triples set
4. else
   (a) For each combination of resource URI on two specified positions of the triple
   (b) Replace the ?subject, ?object, ?predicate in the template
   (c) Process the result and add triples into the resulting triples set
5. return the set of triples

The triples are then shown in the GUI. The triples returned are not just URIs, but they contain all the resource details (textual representations and lemma).

10.2.2 Search for resource

User can test whether or not a resource exists in the database. By specifying any of the triple part or terminology, the application takes this text, creates query to search for a resource URI in the terminology or term or predicate graphs. If a resource has been found, then it extracts all details about this resource and returns them to the user. For terminology the method is *List¡ITerminologyDetail¿ getTerminologyDetail(String terminologyName)* and for part of the triple *List¡IResourceDetail¿ getResourceDetail(String resourceName)*.

10.2.3 Document search based on a triple

The search can be performed by filling all parts of the triple or by any subparts of the triple:

1. Determine which parts of the triple are specified
2. For each part find the resource URIs that has textual representation or lemma same as input text
3. If no resource has been found then return null
4. For each each part and each resource
   (a) Replace the ?subject, ?object, ?predicate string with the resource URI in the database query template *SELECT DISTINCT ?graph WHERE GRAPH ?graph ?subject ?predicate ?object*
   (b) Extract the graph URI from the results and add it into the unique set of graphs
5. For each matched graph extract all details about the graph, triples, terminologies, document file, plain text, statistics
6. return the result

We have decided to extract all information about the matched graph, so the user is able check the result if it suits him and download the original document that has been used for the information extraction.

10.2.4 Document search based on a terminology

On top of the document search based on a terminology, we can provide a way how to sort the result based on the terminology TF-IDF value. TF-IDF value represent how relevant is the word for a particular document, the higher the number is the more relevant is for the document. Therefore we are able to sort the document results based on their relevancy for a given term. The TF-IDF is not stored in the document for each terminology, because the IDF part might change for every new stored document in the database. Algorithm:

1. Find resource with the same textual representation or lemma

2. Create a triple, resource URI, predicate hasFrequency and object is unspecified. Predicate hasFrequency can appear only in document graphs, the resource URI on subject position also in the terminology graph

3. Apply the query and search for the graphs.

4. For each graph extract the details and calculate the TF-IDF for the terminologies

5. Sort the graphs by the TF-IDF of the terminology resource specified in the query

6. Return the graph set

Result is then returned to application GUI.

10.2.5 Document search based on a triple and a terminology

As mentioned in the search requirements, we would like to provide a search based on the triple, but with a more specified constraint based on the terminology. For the given triple is searches for documents having this triple in its graph, then it searches for documents based on the given terminology. Both results are disjointed and only those documents being in both results are returned. The result is then sorted by the TF-IDF value of the terminology.

11. User Interface

In this chapter we will describe the GUI used to interact with the application. We will describe how to process documents and how to search for the information stored in the database.

11.1 Overview

The application GUI can be displayed in any available web browsers due to the features of *Vaadin* framework. The main page is divided into 2 parts, see figure4.1. Section *A* contains controls and allows user to interact with the application, section *B* is used for displaying results.

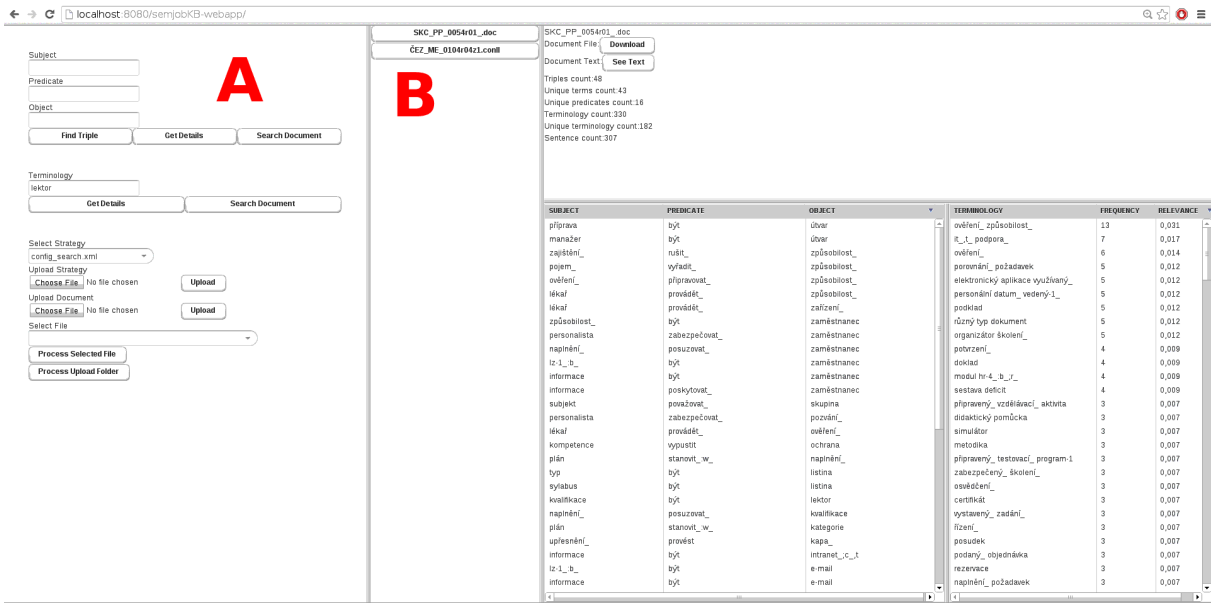


Figure 11.1: Application GUI

11.2 Triples and terminologies

Application operates with informations stored in triples and terminologies. The GUI allows user to insert text into Subject, Predicate and Object textual fields and search for triples stored in the database by clicking the *Find Triple* button. The result will then be displayed in the display section *B*. The result will be displayed in a table by clicking any of the cell in the table a pop up window will be displayed with the information about the resource (resource URI, lemma and textual representations). When user presses *Get Detail* button, the application will search for a resource having the written textual representation or lemma and will display the results. *Search Document* will trigger searching for document matching the inserted triple. Triple can have all parts specified or any parts. The result is shown in figure4.1. However this search has been done on terminology, the result would be the same for the triples. Just over the letter *B* is a list

containing the documents matching the query. By clicking on any item in the list, it will display the information about the document. First line contains the document name. Button *Download* allows to download the document from the server to the client machine. *See Text* button opens the pop-pup windows that contains the plain text of the document. Under these buttons you can find the statistics about the document. Triples, unique terms (subject, object), unique predicate and terminology counts. Such provided information allows user to see how good the IE on this document was. The document result windows contains two tables. One contains the list of triples extracted from the document and the other contains terminologies with their frequencies an TF-IDF value. By clicking on any of the cells a pop-up window will show the resource informations. *Get Details and Search Document* buttons has the same effect for terminologies as for triples. By searching using the terminology, the result document list is sorted by the terminology TF-IDF value in the documents. If we would have filled both, triple parts and terminology, this will trigger searching for document by triple and terminology. The result is then created by disjoin of each result's groups.

11.3 Processing documents

Select strategy combo box contains all IE strategies available on the server, if an user would like to create new IE rules, he has to upload this *XML* file on the server by using the File chooser under *Upload Strategy Label*. File chooser under *Upload Document* label allows user to upload a new document on the server and invoke IE on them. Once he will finish uploading the document on the server, he can select this file in the *Select File* combo box and by pressing the *Process Selected File* button invoke the IE. The application then takes the selected rules strategy file, extracts the text from the document, invokes NLP, extracts knowledge and stores it in the database. On success extracted informations and strategies are displayed in a same way as by searching for the document. However if an user would like to extract informations from multiple documents at once, *Vaadin* framework does not support multiple file transfer. Therefore user needs to upload the files into the `\upload\` folder on the server. And then by using the button *Process Upload Folder*, the applications will start processing each file On success the file is then moved from the upload folder into the `\semjobFiles\` folder where all documents are stored on the server.

11.4 Examples

To demonstrate our application we will insert some examples. Lets start with the document search based on terminology search, see figure11.2. We would like to search for a document (documents) that are about filling a reactor. We will insert the text "*plnění reaktoru*" into the terminology search text field and press *Search Document* button. The result set is consisted from one file, which contains that king o knowledge. In the result tab we can see the document details. In the terminology table we can find our terminology tab on the top of the terminologies sorted by the relevance. We can assume that there is only one document having this kind of knowledge. Because the TF-IDF is high and we did not get any other

results, however the other documents can contain words as "*plnění*" and "*reaktor*" but only in this document these two words were semantically connected.

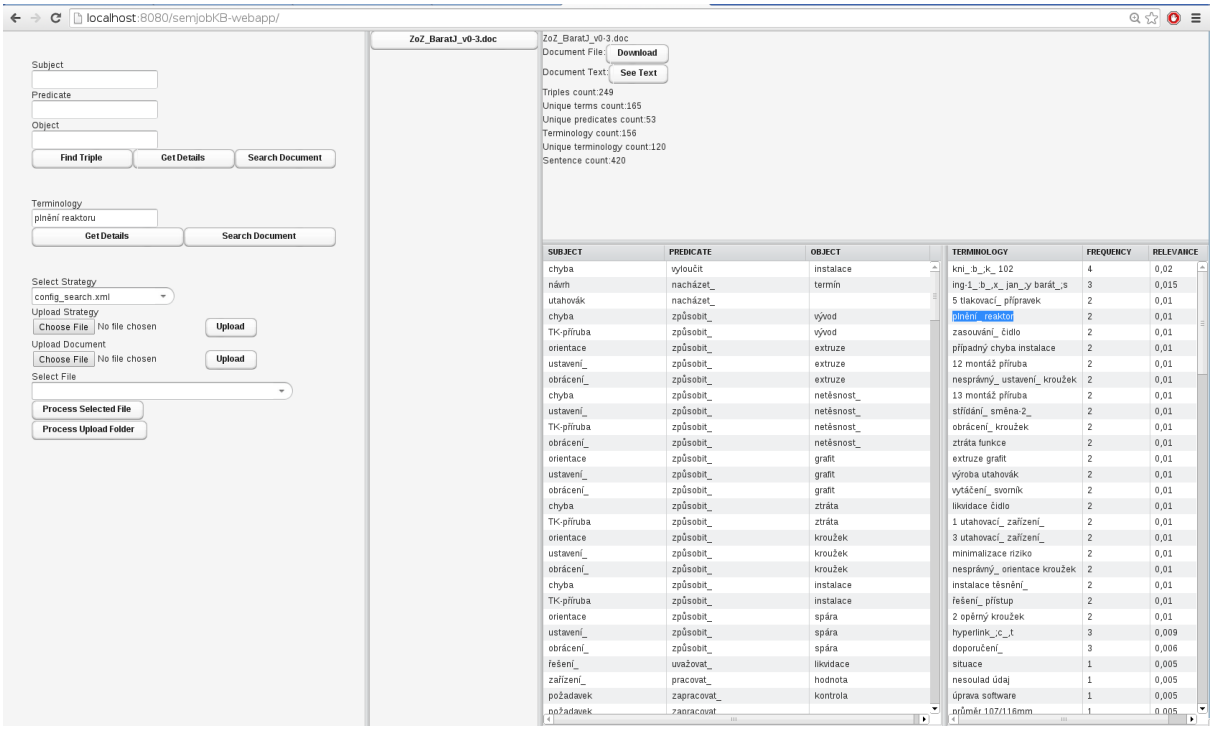


Figure 11.2: Terminology search result

Another example is about finding a triples that has some missing part. We would like to know, in which triples does a subject "*trubička*" appears. See figure11.3. We have found that the subject is presented in 6 triples, by writing the missing parts in the empty search text fields we can execute search for the document.

The last example will show document IE and its results. On the figure11.4 we can see that we uploaded and performed processing on the file "`Program_zastupnictvi_MM_ZP_AKP_2010.doc`" and we have selected "`config_search.xml`" search configuration. The result of the IE is shown on the ride side in the display part of the GUI. The document does not have that many sentences and the number of extracted triples is not high, however we have managed to extract quite a lot of terminologies and their TF-IDF relevance number shows us, that there is a lot of unique terms describing the document.

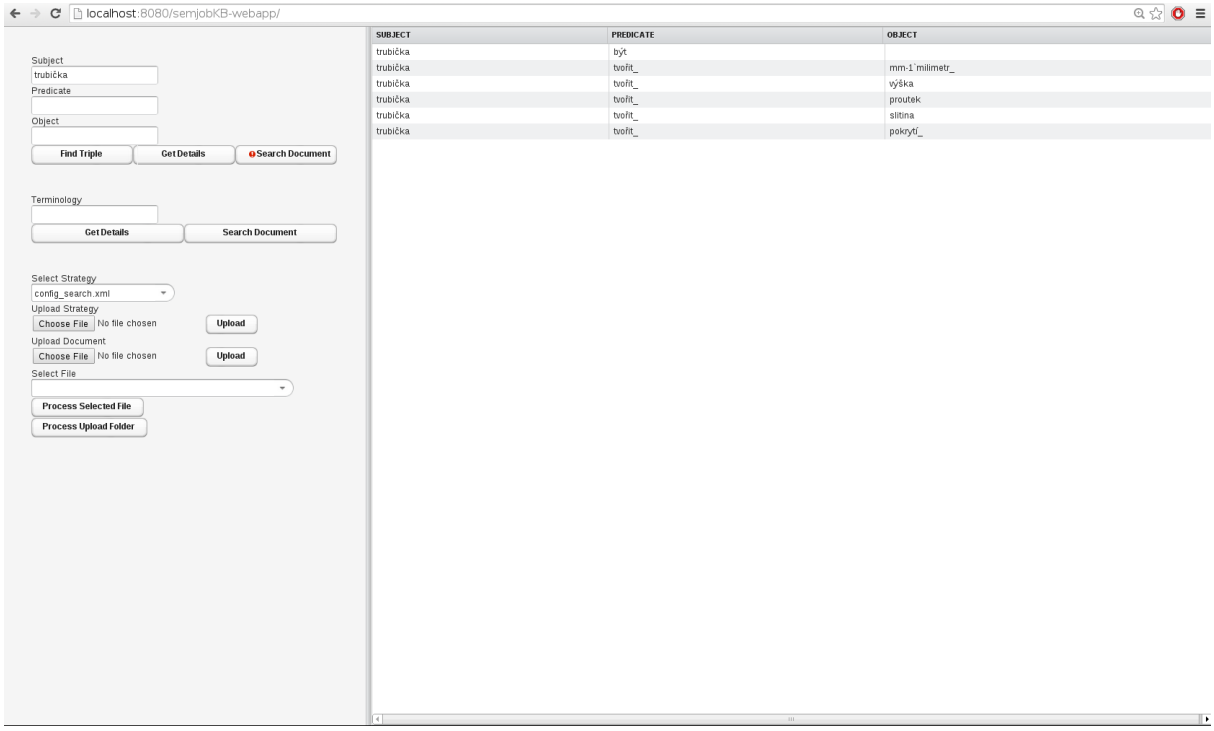


Figure 11.3: Search for missing triple parts

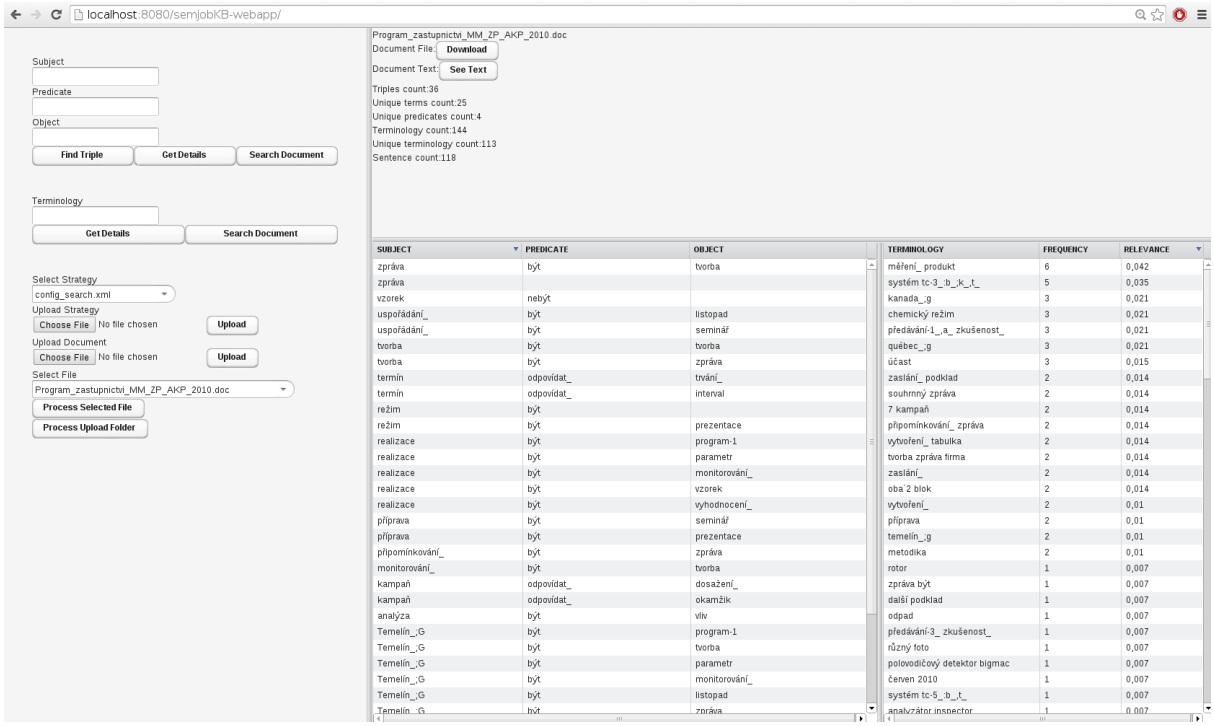


Figure 11.4: Processing a document

12. Results evaluation

In this chapter we will provide evaluations of the IE processed over the testing documents. We will describe the problems we have encountered and possible solutions.

12.1 Results of the thesis

We have applied the search rules on the documents provided to us with following results in a table12.1. The columns are as follow: document group, sentences, sentences with predicate presented, sentences without predicate, words count, triples count, unique terminology count, overall terminology frequency.

We will show now the ratio between the number of sentence and the number of triples and terminologies, see table12.2. First columns contains the group name again, then the ratio between all sentences and number of triples, the ratio between sentences with predicate a triples, the ratio between all sentences and number of unique terminologies and the last column contains the ratio between all sentences and overall terminology frequencies.

From the statistics we can see that except the Management Documentation there is more sentences without predicate then with predicate. We cannot extract triples from sentences without predicate and therefore loosing the opportunity to extract informations in a form of triples. However if sentences have predicates, except the Expert's profiles PowerPoint presentation, we are able to extract almost all triples. The number over 100% means that from these sentences we were able to extract more then 1 triple. We consider full sentence as sentence having the originator (subject), predicate and the target (object). The more of these sentences we would have in the result of NLP the more we can extract from these documents. To determine why there are so many sentences without triples and also the sentences without predicate (phrases) we have to look into the results of the NLP. The PowerPoint presentations have only short phrases and a lot abbreviations. That shows us the fact that full sentences are missing in these presentations and the NLP can not do anything with that. However we have managed to get decent ratio (almost 75%) between the number of sentences and extracted terminologies. Consider this, the terminologies are most suitable to describe the PowerPoint presentations instead of triples. Management documentation, Substitution program and Experience records are Word and PDF (printed from Word) documents with large continuous textual parts. Even if the ratio between triples and all sentences are decent, we have huge amount of sentences without predicate. This might be a problem of NLP or the full sentences are missing there. Looking into the results of NLP and documents we have found that its a mixture

Table 12.1: Information extraction result

Document Group	Sentences	Sent. Pred	Phrases	Words	triples	Un. Termin.	Termin.
Management Documentation	8929	4737	4192	109568	6497	3487	5653
Substitution Program	1138	392	746	12527	464	661	821
Experience records	12405	5738	6667	137586	6042	6225	7920
Expert's profiles	1157	275	882	9675	125	648	867

Table 12.2: Information extraction result

Document Group	Sentences:Triples	SentPred:Triples	Sentences:UniqueTerm	Sentences:Term
Management Documentation	72.76%	137.15%	39.05%	63.31%
Substitution Program	40.77%	118.37%	58.08%	72.14%
Experience records	48.71%	105.30%	50.18%	63.85%
Expert's profiles	10.80%	45.45%	56.01%	74.94%

of both. The NLP tool Treex uses statistical machine learning and models used in the Treex were trained on newspaper articles. The dependency trees of the sentences are considered to be parsed from the full sentences, or sentences close to being them. Any missing part of the sentence or any non grammatically correct sentence leads into wrong or grammatically wrong dependency tree. Even if the humans understand them and can recognize the information which are carried there the machine could not. The only solution is that the person who writes the document will know that the document will be processed by a machine. The person will need to write full sentences and use less abbreviations to increase the efficiency of the NLP. Another great problem with processing the text is, that it contains a lot of information stored in texts indented by bullets. The NLP can not work with informations stored in another sentence that has direct impact in the currently processed sentence. Example:

> *Podkladem pro přiznání a výpočet mimořádné odměny jsou:*
> *- vyhodnocené Prezenční listiny nebo Záznamy o školení (např. v případě školicích dnů, profesních školení, apod.),*
> *- vyplněné Třídní knihy (např. v případě specifické základní přípravy),*
> *- potvrzené Záznamové listy (v případě obecné části stáže),*
> *- protokoly z přezkoušení*

Treex is unable to process this structure and mark full sentences from them, because text is fragmented into individual sentences. The above example can be rewrite into a model:

- **A:**
  - **B**
  - **C**
  - **D**
  - **E**

The possible solution could be to reorganized the example by using regular expression and create sentences as A B, A C, A D, A E. However are we sure that the model is uniform across all documents and only A contains the predicate, subject and B,C,D,E only objects? We have tried different strategies, increasing number of full sentences for particular part of the text and in the same time decreasing number of full sentences in the same sentences. Without uniting the style of use of bullets we cant create a general strategy for solving this issue.

> *Lektor – zaměstnanec ČEZ, a. s., který na základě písemného pověření (Zadání lektorské činnosti) vystaveného útvarem RLZ realizuje odbornou problematiku formou teoretického školení určenému okruhu zaměstnanců.*

This example shows that there is a missing predicate in the sentence, the predicate is replaced with a dash. Treex thinks that the dash is a delimiter and splits this example into two separate sentences. We do not even loose the sentence structure but also we loose the subject. To fix this problem we would need to replace the dash with the right predicate, in this example with verb is. The problem seems unsolvable at the moment due to the fact, that not even we have to choose the appropriate verb but also the correct inflexion of the verb. And the inflexion is rich in the Czech language. Another aspect of the example marks another fact. For NLP is hard to process very large clauses, that are consisted from main and subordinate clauses. The chance of hitting the right dependency tree is getting smaller with the increasing number of possible variations of the clause's dependency trees. The more simple the clause is the bigger chance of getting the right dependency tree is. If we could fix or at least decrease the number of sentences in which we are loosing possible extracted informations, we could achieve higher number of extracted triples and terminologies. Not only higher number of extracted informations but we could also covered larger parts of the document. Seeing this problems still we were able to extract a solid number of informations to work with. We have extracted text from various newspaper articles. The text was mostly consisted from large pieces of continuous texts. The Treex returned 85% sentences with predicate and 15% were phrases. Resulting IE achieved sentence triples ratio over 80%. This implies that having a grammatically correct with continuous text documents can provides sufficient base for NLP and IE.

12.2 Summary

During implementation and testing of the IE on the documents, we have encountered significant losses during the NLP of the texts. This problems have been described and possible solutions proposed. The performance for extracting informations is sufficient, the only bottleneck here is the performance of Treex. With the extracted information we can advance to the next chapter.

Conclusion

In the thesis we have explored the possibility of IE from unstructured documents provided by CEZ, a.s.. Trough a research and observations we have created an application that will allow user to extract informations, store them in a database and search over them. The extraction is based on pattern rules that we have invented or users can create and apply their own. Users can interact with the application trough a web based GUI. The software was designed to be extensible by using decomposition of the problems into individual tasks that are implemented in modules. The modules are implementing well defined interfaces and can be altered without any significant impacts on the rest of the application. All goals of this thesis were analyzed and possible solutions described. However we can not be satisfied with the results. Even that we were able to extract a lot of s in a form of triples and terminologies. For the machines the results are just data. For humans, the results of IE have to carry a meaning. Some of the triples and terminologies that were matched by the well designed patterns based on the NLP research simply did not make sense from a human perspective. Therefore are useless for searching, because human would never search for a triple or terminology that does not make sense (carries information). Also the overall number of terminologies and triples is not high enough. We were successful overall only in half of the sentences. We have explained that it is because the NLP tool Treex is not trained for such documents and also the documents contain sentences which are grammatically incorrect causing confusion for the Treex. However these are real data and we have to work with them, learn from them and improve and specify for which kind of knowledge are we interested in. Consider the knowledge relevancy between the document and extracted triples and terminologies, the precision describing informations stored in documents goes in favor of terminologies. Based on the advancement of NLP research and the real life data, the IE of triples is still too difficult and results are imprecise. For future development in the knowledge domain extraction we will recommend focusing on terminologies extraction. Current terminology extraction works only with a frequency of the words in the language or specific domain and a frequency found in documents. Combining the NLP results together with their frequencies provides a good way how to catch the knowledge stored in documents and offers more relevant and more precise search results then current full-text search engines. Apart of the IE we have interconnected business documents with the semantic database and NLP tool. We have discovered that in current state the NLP tool Treex is not appropriate for business use due to the poor performance when processing large number of documents. We have also found that the semantic databases are ideal for tasks such as knowledge extraction and manipulating with that kind of data. If we could improve the quality of the texts in the documents together with training the NLP for this kind of business documentation we would have get much more relevant informations in the future. This is the main goal for the future work in this field of computer science. If we could manage all of this, it would definitely lead into new era of Document Management Systems for business companies.

Bibliography

[1] SINGHAL, Amit. "Introducing the Knowledge Graph: things, not strings." *Google blogspot* 2012. Web. <http://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html>

[2] <http://linkeddata.org>

[3] Jiang, Jing. "Information Extraction from Text." Mining Text Data. Boston: Springer, 2012. 11-41. Web. <http://link.springer.com/chapter/10.1007/978-1-4614-3223-4_2>.

[4] Andersen, Peggy M.; Hayes, Philip J.; Huettner, Alison K.; Schmandt, Linda M.; Nirenburg, Irene B.; Weinstein, Steven P. "Automatic Extraction of Facts from Press Releases to Generate News Stories". CiteSeerX, 1992. Web.<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.7943>

[5] Cowie, Jim; Wilks, Yorick. "Information Extraction". CiteSeerX, 1996 Web. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.6480>

[6] Cowie, Jim; Lehnert, Wendy. "Communications of the ACM: Information Extraction". ACM New York, 1996, Volume 39, Issue 1, DOI 10.1145/234173.234209 Web. <http://dl.acm.org/ft_gateway.cfm?id=234209&ftid=27245&dwn=1&CFID=348210643&CFTOKEN=27446945>

[7] Wendy Lehnert, Claire Cardie, Divid Fisher, Ellen Riloff, and Robert Williams. University of Massachusetts: Description of the CIRCUS system as used for MUC-3. In Proceedings of the 3rd Message Understadning Conference, pages 223–233, 1991.

[8] Ralph Grishman, John Sterling, and Catherine Macleod. New York University: Description of the PROTEUS system as used for MUC-3. In Proceedings of the 3rd Message Understadning Conference, pages 183–190, 1991.

[9] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the Web. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, pages 2670–2676, 2007.

[10] Fei Wu and Daniel S. Weld. Open information extraction using Wikipedia. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 118–127, 2010.

[11] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In Proceedings

of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 1535–1545, 2011.

[12] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the Web. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, pages 2670–2676, 2007.

[13] Automatic content extraction program. (ACE) http://www.itl.nist.gov/iad/mig/tests/ace/

[14] Popel Martin, Žabokrtský Zdeněk: TectoMT: Modular NLP Framework. In: Lecture Notes in Computer Science, Vol. 6233, Proceedings of the 7th International Conference on Advances in Natural Language Processing (IceTAL 2010), Copyright ? Springer, Berlin / Heidelberg, ISBN 978-3-642-14769-2, ISSN 0302-9743, pp. 293-304, 2010

[15] Menczer F., Pant G. and Srinivasan P. Topic-Driven Crawlers: machine learning issues, in ACM Transactions on Information Systems (TOIS), 2002

[16] Yan Zheng Wei, Luc Moreau, Nicholas R. Jennings. A market-based approach to recommender systems, in ACM Transactions on Information Systems (TOIS), 23(3), 2005.

[17] Yangarber, R. (2003). Counter-training in the Discovery of Semantic Patterns. In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03), Sapporo, Japan, pp. 343–350.

[18] STEVENSON, Mark a Mark A. GREENWOOD. "Research on Language and Computation: Dependency Pattern Models for Information Extraction. Springer Netherlands, 2009, Volume 7, Issue 1. ISSN 1572-8706. Web. <http://link.springer.com/article/10.1007%2Fs11168-009-9061-2>

[19] http://en.wikipedia.org/wiki/Natural_language_processing

[20] http://www.w3.org/RDF/

[21] http://ufal.mff.cuni.cz/pdt2.0/index-cz.html

[22] http://ufal.mff.cuni.cz/pdt2.0/doc/pdt-guide/cz/html/ch02.html

[23] Zeman, D., Hana, J., Hanová, H., Hajič, J., Hladká B., Jeřábek, E.: A Manual for Morphological Annotation, 2nd ed., Technical Report 27, UFAL MFF UK, Prague, Czech Republic (2005)

[24] Hajičová, E., Kirschner, Z., Sgall, P.: A Manual for Analytic Layer Annotation of the Prague Dependency Treebank (English translation). Technical report, UFAL MFF UK, Prague, Czech Republic (1999)

[25] Mikulová, M., Bémová, A., Hajič, J., Hajičová, E., Havelka, J., Kolářová, V., Kučová, L., Lopatková, M., Pajas, P., Panevová, J., Razímová, M., Sgall, P., Štěpánek, J., Urešová, Z., Veselá, K., Žabokrtský, Z.: Annotation on the tectogrammatical level in the Prague Dependency Treebank. Annotation manual. Technical Report 30, UFAL MFF UK, Prague, Czech Rep (2006)

[26] http://ufal.mff.cuni.cz/treex/index.html

[27] http://opennlp.apache.org/

[28] http://nlp.stanford.edu/downloads/corenlp.shtml

[29] http://ufal.mff.cuni.cz/tools.html/morce.html

[30] http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html

[31] http://ilk.uvt.nl/conll/

[32] http://ilk.uvt.nl/conll/#dataformat

[33] http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Doc/hmptagqr.html

[34] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s02.html

[35] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s03.html

[36] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s04.html

[37] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s05.html

[38] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s06.html

[39] http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s07.html

[40] http://www.linkeddatatools.com/introducing-rdf

[41] http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/rdfsemantic-graph-1902016.html

[42] http://www.openrdf.org

[43] http://virtuoso.openlinksw.com

[44] http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html

[45] http://www.w3.org/TR/rdf-sparql-query/

[46] http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[47] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtJenaProvider

List of Tables

List of Figures

List of Abbreviations

NLP - Natural Language Processing
IE - Information Extraction
NER - Named Entity Recognition
RE - Relation Extraction
PDF - Portable Document Format
NP - Noun Phrases
RDF - Resource Description Framework
GUI - Graphic User Interface
GWT - Google Web Toolkit
RMI - Remote Method Invocation
TF-IDF - Term Frequency - Inverse Document Frequency

A. Appendix - Treex Installation

The Treex installation guide can be found on `http://ufal.mff.cuni.cz/treex/install.html`. This tool cannot be installed trough standard installer or package installer. The only supported platform is Linux.The user should have basic knowledge of shell, Linux package installer and Perl module installer. The first step consists of setting a user environment for Perl. After the environment setup is done, user has to restart the terminal apply the changes. Treex requires several Perl modules to be installed. The guide uses cpanm application to install the required modules. However if the user does not have the cpanm he has to install it first to continue with the installation of Treex. If the installation of the Perl modules fails, the user has to read the error messages and install required development dependencies using package installer in Linux. However as mentioned in the guide, some Perl modules are failing in newer version of Perl(The exception is a problem with UNIVERSAL::DOES and PerlIO::Util, which won't get installed without –notest on the newest Perl 5.18).

Missing packages, modules:

- *libxml2-devel*
- *zlib1g-devel*
- *perl-YAML*
- perl module *File::ShareDir::Install, POE, App::whichpm*

The user can install an optional application TrEd. TrEd is a fully customizable and programmable graphical editor and viewer for tree-like structures and its used to view and edit the outputs of the linguistic processing that are done by Treex.

After installing the Treex, there is a brief introduction site for using Treex `http://ufal.mff.cuni.cz/treex/firststeps.html`. User should try to run every command to test if every module is installed and read, resolve every error message. The first error was a missing module *tagset common.pm*. I could not found this module on cpanm and therefore could not install it directly. I have found the module on `https://wiki.ufal.ms.mff.cuni.cz/user:zeman:interset`. From there I have checkout the project and added it into the the *treex/lib* folder. Also other missing modules could not be installed from cpanm and had to be manually downloaded from SVN repository `https://svn.ms.mff.cuni.cz/projects/tectomt_devel/browser/trunk/libs/other`. When running tokenizer, tagger and parser the models are not part of the svn checkout of latest Treex version, we need to download models from `http://ufallab.ms.mff.cuni.cz/tectomt/share/data/models/` that contains models as CzechMorpho and MST parsers. These models need to be added into $*HOME/treex/share/data/models/*. For example *CzechMorpho* is stored in `http://ufallab.ms.mff.cuni.cz/tectomt/share/data/models/morpho_analysis/cs/` and has to be downloaded into `$HOME/treex/share/data/models/morpho_analysis/cs/`. In the same way we need to install other missing models if there are some left.

B. Appendix - Ontology

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix semjobKB: <http://datlowe.org/semjobKB/ontologies/domain/document#>.
@base <http://datlowe.org/semjobKB/ontologies/configuration#>.


# Document ontology
semjobKB: a owl:Ontology.

###############
# Document
###############

# Classes
semjobKB:Document a owl:Class .

semjobKB:hasName a owl:ObjectProperty ;
  rdfs:domain semjobKB:Document ;
  rdfs:range xsd:string .

semjobKB:hasLocation a owl:ObjectProperty ;
  rdfs:domain semjobKB:Document ;
  rdfs:range xsd:string .


###############
# Document parts
###############

semjobKB:Term a owl:Class .

semjobKB:Predicate a owl:Class .

semjobKB:TerminologyItem a owl:Class .

semjobKB:has a owl:ObjectProperty ;
  rdfs:domain semjobKB:Document ;
  rdfs:range semjobKB:TerminologyItem, semjobKB:Term, semjobKB:Predicate.

semjobKB:Predicate a owl:ObjectProperty ;
  rdfs:domain semjobKB:Term ;
  rdfs:range semjobKB:Term .

semjobKB:language a owl:ObjectProperty ;
  rdfs:domain semjobKB:Term, semjobKB:Predicate, semjobKB:TerminologyItem, semjobKB:Document;
  rdfs:range xsd:string .

semjobKB:hasLemma a owl:ObjectProperty ;
  rdfs:domain semjobKB:Term, semjobKB:Predicate, semjobKB:TerminologyItem ;
  rdfs:range xsd:string .

semjobKB:hasStringRepresentation a owl:ObjectProperty ;
  rdfs:domain semjobKB:Term, semjobKB:Predicate, semjobKB:TerminologyItem ;
  rdfs:range xsd:string .


##############
# Terminology
##############

semjobKB:hasFrequency a owl:ObjectProperty ;
  rdfs:domain semjobKB:Terminology ;
  rdfs:range xsd:integer .

```

```
67 semjobKB:hasOverallFrequency a owl:ObjectProperty ;
68   rdfs:domain semjobKB:Terminology ;
69   rdfs:range xsd:integer .
70
71 semjobKB:hasTotalCount a owl:ObjectProperty ;
72   rdfs:domain semjobKB:TerminologyCounter ;
73   rdfs:range xsd:integer .
74
75 ###############
76 # Counters on Document
77 ###############
78
79 semjobKB:hasTermCount a owl:ObjectProperty ;
80   rdfs:domain semjobKB:DocumentStatistic ;
81   rdfs:range xsd:integer .
82
83 semjobKB:hasPredicateCount a owl:ObjectProperty ;
84   rdfs:domain semjobKB:DocumentStatistic ;
85   rdfs:range xsd:integer .
86
87 semjobKB:hasTerminologyCount a owl:ObjectProperty ;
88   rdfs:domain semjobKB:DocumentStatistic ;
89   rdfs:range xsd:integer .
90
91 semjobKB:hasSentenceCount a owl:ObjectProperty ;
92   rdfs:domain semjobKB:DocumentStatistic ;
93   rdfs:range xsd:integer .
```

C. Appendix - Search configuration example

```
<?xml version="1.0" encoding="UTF-8"?>
<search>
  <ruleset type="predicate">
    <rule>
      <node>
        <constituent>PREDICATE</constituent>
        <vassal>false</vassal>
        <parentType>null</parentType>
        <wordType>VERB</wordType>
      </node>
    </rule>
    <rule>
      <node>
        <constituent>PREDICATE</constituent>
        <vassal>false</vassal>
        <parentType>null</parentType>
        <wordType>VERB</wordType>
      </node>
      <node>
        <constituent>AUXILIARY_VERB</constituent>
        <vassal>true</vassal>
        <parentType>PREDICATE</parentType>
        <wordType>null</wordType>
      </node>
    </rule>
    <rule>
      <node>
        <constituent>PREDICATE</constituent>
        <vassal>false</vassal>
        <parentType>null</parentType>
        <wordType>VERB</wordType>
      </node>
      <node>
        <constituent>PREDICATE_NOMINAL</constituent>
        <vassal>true</vassal>
        <parentType>PREDICATE</parentType>
        <wordType>null</wordType>
      </node>
    </rule>
  </ruleset>
  <ruleset type="object">
    <rule>
      <node>
        <constituent>PREDICATE</constituent>
        <vassal>false</vassal>
        <parentType>null</parentType>
        <wordType>VERB</wordType>
      </node>
      <node>
        <constituent>OBJECT</constituent>
        <vassal>false</vassal>
        <parentType>PREDICATE</parentType>
```

```
53          <wordType>NOUN</wordType>
54        </node>
55      </rule>
56      <rule>
57        <node>
58          <constituent>PREDICATE</constituent>
59          <vassal>false</vassal>
60          <parentType>null</parentType>
61          <wordType>VERB</wordType>
62        </node>
63        <node>
64          <constituent>ATRIBUT</constituent>
65          <vassal>false</vassal>
66          <parentType>PREDICATE</parentType>
67          <wordType>NOUN</wordType>
68        </node>
69      </rule>
70    </ruleset>
71    <ruleset type="subject">
72    <rule>
73        <node>
74          <constituent>PREDICATE</constituent>
75          <vassal>false</vassal>
76          <parentType>null</parentType>
77          <wordType>VERB</wordType>
78        </node>
79        <node>
80          <constituent>SUBJECT</constituent>
81          <vassal>false</vassal>
82          <parentType>PREDICATE</parentType>
83          <wordType>NOUN</wordType>
84        </node>
85      </rule>
86    </ruleset>
87    <ruleset type="terminology">
88      <rule>
89        <node>
90          <constituent>EX_DEPENDENT</constituent>
91          <vassal>false</vassal>
92          <parentType>null</parentType>
93          <wordType>NOUN</wordType>
94        </node>
95        <node>
96          <constituent>ATRIBUT</constituent>
97          <vassal>true</vassal>
98          <parentType>EX_DEPENDENT</parentType>
99          <wordType>null</wordType>
100       </node>
101     </rule>
102     <rule>
103       <node>
104         <constituent>SUBJECT</constituent>
105         <vassal>false</vassal>
106         <parentType>null</parentType>
107         <wordType>NOUN</wordType>
108       </node>
109       <node>
110         <constituent>ATRIBUT</constituent>
```

```
111           <vassal>true</vassal>
112           <parentType>SUBJECT</parentType>
113           <wordType>null</wordType>
114         </node>
115       </rule>
116       <rule>
117         <node>
118           <constituent>OBJECT</constituent>
119           <vassal>false</vassal>
120           <parentType>null</parentType>
121           <wordType>NOUN</wordType>
122         </node>
123         <node>
124           <constituent>ATRIBUT</constituent>
125           <vassal>true</vassal>
126           <parentType>OBJECT</parentType>
127           <wordType>null</wordType>
128         </node>
129       </rule>
130     </ruleset>
131 </search>
```